С. Хашими | С. Коматинени | Д. Маклин



РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ Android

Apress®

ПИТЕР®


ПИТЕР®

S. Hashimi | S. Komatineni | D. MacLean

Pro Android 2

Apress®

С. Хашими | С. Коматинени | Д. Маклин

РАЗРАБОТКА ПРИЛОЖЕНИЙ ДЛЯ Android



Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Ростов-на-Дону · Екатеринбург · Самара · Новосибирск
Киев · Харьков · Минск
2011

ББК 32.973.2-018+32.882.9
УДК 004.451
Х2

Хашими С., Коматинени С., Маклин Д.
Х2 Разработка приложений для Android. — СПб.: Питер, 2011. — 736 с.: ил.

ISBN 978-5-459-00530-1

Благодаря этому практическому руководству вы научитесь создавать приложения для устройств на базе ОС Android (мобильных телефонов, планшетных компьютеров, нетбуков, смартбуков), пользуясь новейшими инструментами разработки. Помимо основных вопросов и методик написания программ для Android, в книге рассмотрены более сложные темы, в частности, создание пользовательских 3D-компонентов, работа с OpenGL и сенсорными экранами, в том числе обработка жестов. Вы узнаете об интегрированных в Android функциях локального и глобального поиска, о внедрении функции машинного перевода Google, о функциях синтеза речи.

Кроме подробного теоретического материала, в книге содержатся практические рекомендации от профессионалов и примеры готовых работающих приложений. Вы получите все необходимые знания и навыки для написания приложений любой сложности!

ББК 32.973.2-018+32.882.9
УДК 004.451

ISBN 978-1430226598 англ.
ISBN 978-5-459-00530-1

Краткое содержание

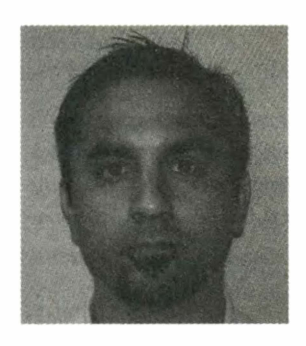
Оглавление

Моему сыну Сейду-Адибу.
Сейд Хашими

Моей прекрасной жене Энн-Мэри за ее дух; Эшли за ее неисчерпаемую надежду; Николасу за его добродушие; Кавитхе за острый ум и великолепные способности; Нараян за непревзойденную красоту и всей моей большой семье в Индии и США за их любовь.
Сатья Коматинени

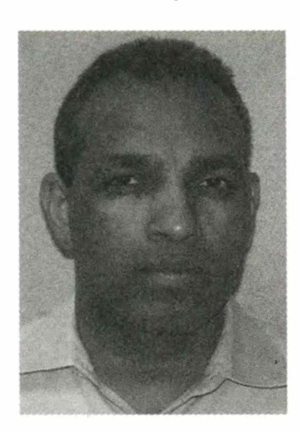
Моей жене Рози и моему сыну Майку за их поддержку; без них я не смог бы написать этот труд. И Максу за то, что провел так много времени со мной, составляя мне компанию.
Дэйв Маклин

Об авторах

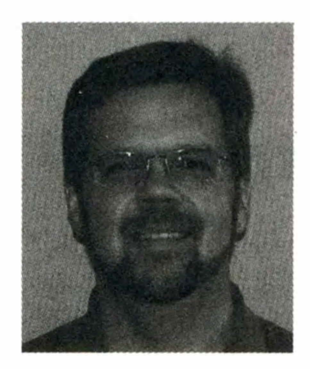


Сейд Хашими (Sayed Y. Hashimi) родился в Афганистане; сейчас живет в Джэксонвилле, штат Флорида (США). Он работал в сферах здравоохранения, финансов, логистики и сервис-ориентированной архитектуры. В ходе своей карьеры Сейд разрабатывал крупномасштабные распределенные приложения на нескольких языках программирования и платформах, в том числе на C/C++, MFC, J2EE и .NET. Его статьи публиковались в крупных научных журналах по программированию. Кроме того, он является автором нескольких популярных книг, вышедших в издательстве Apress. Сейд имеет степень магистра технических наук в университете Флориды. Вы можете связаться с Сейдом, посетив его сайт — www.sayedhashimi.com[1].

Сатья Коматинени (Satya Komatineni) (www.satyakomatineni.com) занимается программированием более 20 лет, работал с крупными и небольшими компаниями. Сатья написал более 30 статей, посвященных веб-разработке с использованием Java, .NET и технологий баз данных. Он часто выступает на промышленных конференциях, посвященных инновационным технологиям, и регулярно пишет в блогах на сайте java.net. Сатья является автором AspireWeb (www.activeintellect.com/aspire), упрощенного свободно распространяемого инструмента для веб-разработки на Java, и создателем Aspire Knowledge Central (www.knowledgefolders.com), свободно распространяемой операционной системы (ОС) для веб, в которой сделан акцент на функциях индивидуальной продуктивности и публикации материалов. Кроме того, Сатья является членом-спонсором некоторых программ по изучению рационализаторских предложений от небольших фирм (Small Business Innovation Research Programs, SBIR). Он получил степень бакалавра по электротехнике в университете Андхра, город Вишакхапатнам (Индия), и степень магистра в области электротехники



[1] На момент подготовки русскоязычного издания данный интернет-адрес (а также некоторые другие, приведенные в книге далее) не работал. Однако мы все равно оставляем адреса, указанные в оригинале, надеясь на то, что к выходу книги они снова будут доступны. *(Примеч. ред.)*



в Индийском технологическом институте, город Нью-Дели.

Дэйв Маклин (Dave MacLean) — программист и архитектор программ, в настоящее время живет и работает в городе Джэксонвилле, штат Флорида (США). С 1980 года занимается программированием, работал со многими языками и разрабатывал различные продукты — от систем роботизации до программ для объединения баз данных, независимых веб-приложений и процессоров для EDI-транзакций. Дэйв сотрудничал с компаниями Sun Microsystems, IBM, Trimble Navigation, General Motors и некоторыми небольшими компаниями. Он окончил университет Ватерлоо в Канаде, получив степень в области программирования дизайна систем. Более подробную информацию вы можете узнать на нашем сайте — http://www.androidbook.com.

О техническом редакторе



Викрам Гоял (Vikram Goyal) — программист-разработчик, живет в Брисбене, Австралия. Он взял небольшой отпуск, чтобы отдохнуть со своими детьми. Вы можете связаться с ним по адресу vikram@craftbits.com.

Благодарности

Чтобы эта книга вышла в свет, потребовались усилия не только со стороны авторов, но и со стороны исключительно талантливых сотрудников издательства Apress, и со стороны технического редактора. Поэтому мы хотели бы поблагодарить Стива Энглина (Steve Anglin), Дугласа Пандика (Douglas Pundick), Фрэна Пэрнелла (Fran Parnell), Элизабет Бери (Elizabeth Berry) и Бриджид Даффи (Brigid Duffy) из издательства Apress. Кроме того, мы хотели бы выразить признательность нашему техническому редактору Викраму Гоялу за его вклад в работу над книгой. Его замечания и исправления невозможно переоценить. Наконец, авторы глубоко благодарны своим семьям за терпение и понимание.

Предисловие

Думай. Набирай код. Пиши. Умойся и продолжай так до бесконечности. Это мантра технического писателя. Технологии меняются так быстро, что, когда автор поставит точку в последнем предложении, настанет время переписывать книгу. Вы как читатель технической литературы, вероятно, уже знаете об этом, и все же вы решили купить эту книгу и прочитать ее. Более того, вы даже нашли время, чтобы прочитать это предисловие. Это означает, что вы не просто кодировщик, подрабатывающий по ночам, а читатель, который желает разбираться не только в технологии, но и в метатехнологии. Итак, хорошо — поздравляем вас с тем, что вы потратили часть своих денег на эту книгу. Позвольте нам убедить вас, что вы приняли правильное решение, купив это пособие.

Это лучшая книга об Android, которая есть сегодня в продаже. В ней так много глав, посвященных ценнейшим функциям Android, что вы неоднократно поблагодарите себя за решение приобрести этот труд. Я — технический редактор этой книги. И, честно говоря, мне пришлось заниматься в основном стилистическими правками — авторы отлично сделали свое дело. Мне практически ничего не потребовалось исправлять. Правда, пару раз я ругнулся на них за то, сколько информации они умудрились уложить под обложку, чем подбросили мне большой кусок напряженной работы. Но мои хлопоты — это выгода для вас: в этой книге рассмотрены решительно все вопросы, которые вам потребуется изучить для работы с Android. Взляните на оглавление, чтобы убедиться в этом.

Сложилась традиция, в соответствии с которой я должен сказать пару слов об Android — теме этой книги. Конечно, вы уже что-то слышали о ней: Android — это операционная система от Google, которая, по мнению разработчиков, может побороться с iPhone за лидерство на рынке. Технология Android выросла в тени конкурентов, но теперь с ней нужно считаться, так как на рынок выпущен NexusOne — телефон от Google, работающий под управлением Android. В мире хватит места, чтобы обе технологии сосуществовали друг с другом, но, учитывая мощные позиции Google в Интернете, сотрудникам Apple придется понервничать.

Итак, вы уже сделали два шага: а) решили заняться разработкой под Android и б) купили лучшую книгу по этой теме, чтобы изучить Android. Теперь осталось сделать последний шаг — перевернуть страницу и углубиться в чтение.

Викрам Гоял,
vikram@craftbits.com,
www.craftbits.com.
Январь 2010 года.
Брисбен, Австралия

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу электронной почты halickaya@minsk.piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства http://www.piter.com вы найдете подробную информацию о наших книгах.

1 Введение в компьютерную платформу Android

Компьютеры становятся все более «персональными», возможности доступа к ним в любое время и из любого места все время расширяются. В авангарде этого процесса находятся мобильные устройства, которые трансформируются в компьютерные платформы. Мобильные телефоны уже давно используются не только для разговоров — с их помощью можно за определенный период времени передать данные и видео. Мобильные устройства стали выполнять такой широкий спектр компьютерных задач общего профиля, что именно такие устройства могут стать новым поколением персональных компьютеров (ПК). Кроме того, даже ожидается, что некоторые производители традиционных моделей ПК — в частности, ASUS, HP и Dell — будут делать устройства, многие конструктивные параметры которых будут основаны на ОС Android. Фронт, на котором разворачивается борьба между операционными системами, вычислительными платформами, языками программирования и средами разработки, смещается в сторону мобильных устройств.

Скоро в IT-индустрии ожидается быстрое наращивание темпов и объемов программирования для мобильных устройств, так как у все большего количества приложений появляются мобильные аналоги. Чтобы вы не стояли в стороне от этой тенденции, мы расскажем, как применять Java для написания программ, работающих на платформе Google Android (http://developer.android.com/index.html), свободной платформе для разработки мобильных приложений. Мы воодушевлены потенциалом Android, поскольку это очень продвинутая платформа, на которой применяется ряд новых парадигм дизайна фреймворка (даже с учетом ограничений, с которыми сопряжена работа на мобильной платформе).

В этой главе будет сделан обзор Android и его комплекта для разработки программ (SDK). Мы рассмотрим основные пакеты, кратко расскажем, каким темам будут посвящены следующие главы, объясним, как можно извлечь пользу из исходного кода Android, и особо подчеркнем преимущества программирования для платформы Android.

Новая платформа для нового персонального компьютера

Тот факт, что выделенные устройства, такие как мобильные телефоны, станут в один ряд с другими маститыми платформами общего назначения (рис. 1.1), не-

мало озадачивает программистов. Эта новая тенденция открывает доступ к мобильным устройствам для традиционных языков программирования, благодаря чему диапазон применения мобильных приложений и их доля на рынке растут.



Рис. 1.1. Мобильное устройство — это новый ПК

Платформа Android реализует идею адаптации программ общего назначения к мобильным устройствам. Это многосторонняя платформа, представляющая собой программный стек операционной системы на основе Linux, предназначенный для управления устройствами, памятью и процессами. В библиотеках Android содержатся функции, связанные с телефонией, видео, графикой, программированием пользовательских интерфейсов и некоторыми другими возможностями мобильного устройства.

ПРИМЕЧАНИЕ

Хотя платформа Android и предназначена для разработки под мобильные устройства, она обладает качествами полноценного фреймворка для локального компьютера. Google позволяет писать для этого фреймворка программы на Java, предоставляя разработчикам комплект инструментов (Software Development Kit) под названием Android SDK. Если вы работаете с этим инструментарием, то у вас почти не возникает ощущения, что вы создаете программу для мобильного устройства, так как вам открыт доступ к большинству библиотек классов, используемых на локальном компьютере или сервере, в том числе к реляционной базе данных.

Комплект Android SDK поддерживает большинство функций платформы Java Standard Edition (Java SE), кроме абстрактного оконного интерфейса (Abstract Window Toolkit, AWT) и Swing. Вместо AWT и Swing в Android применяется собственный *расширенный современный фреймворк пользовательского интерфейса*. Поскольку вы пишете приложения на Java, вам может понадобиться виртуальная машина Java (Java Virtual Machine, JVM), которая отвечает за интерпретацию исполняемого байт-кода Java. Обычно JVM обеспечивает необходимую оптимизацию, чтобы помочь Java достичь уровней производительности, сравнимых с аналогичными уровнями транслируемых языков — таких как C и C++. В Android предлагается собственный оптимизированный вариант JVM для исполнения скомпилированных файлов классов Java. Это делается, чтобы преодолеть ограничения, которые

свойственны для мобильных устройств, в частности связанные с памятью, скоростью работы процессора и мощностью. Эта виртуальная машина называется Dalvik VM. Мы подробно рассмотрим ее в разделе «Глубокий анализ Dalvik VM» этой главы.

Понятность и простота языка Java, усиленная обширной библиотекой классов Android, превращает Android в конкурентоспособную платформу для написания программ.

На рис. 1.2 сделан обзор программного стека Android (мы рассмотрим его подробнее в разделе «Программный стек Android» этой главы).

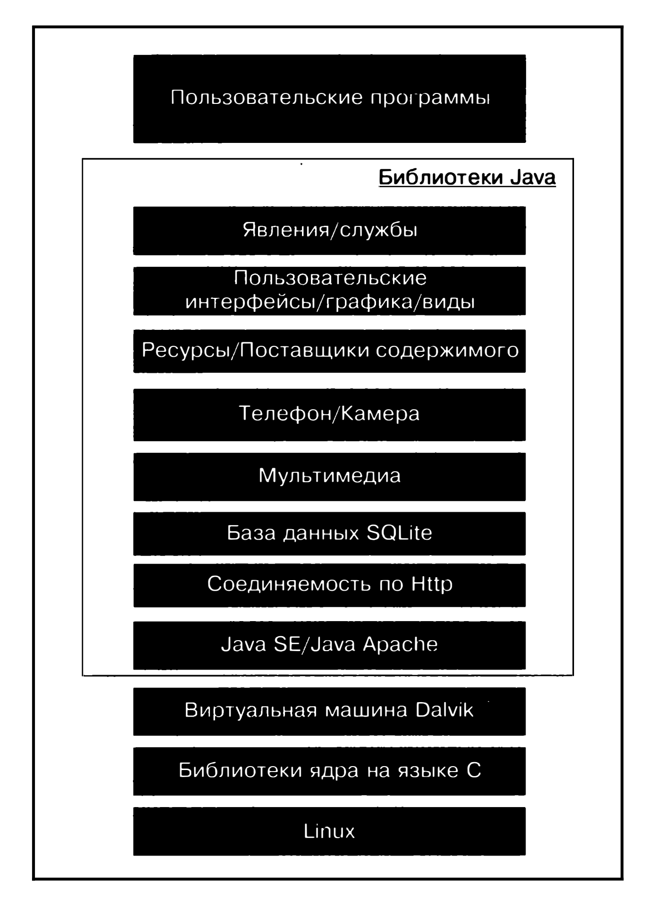


Рис. 1.2. Схематическое представление программного стека Android

История Android

Как Android появился в сфере операционных систем для мобильных устройств? В мобильных телефонах используется целый ряд операционных систем — например, Symbian OS, Microsoft Windows Mobile, Mobile Linux, iPhone OS (на базе Mac OS X),

Moblin (от Intel) и большое количество других патентованных операционных систем. Но пока еще ни одна из ОС не стала стандартом де-факто. Современные интерфейсы прикладного программирования (API) и среды для разработки мобильных приложений имеют слишком много ограничений и, кажется, сильно уступают аналогичным фреймворкам для локальных компьютеров. И вот в этой сфере появляется Google. Платформа Android выглядит многообещающе — для нее характерны и открытость, и доступность, ее код распространяется свободно, а фреймворк разработки отличается высоким техническим уровнем.

Google приобрела стартап Android Inc. в 2005 году, чтобы приступить к разработке платформы Android (рис. 1.3). Ведущими фигурами в Android Inc. были Энди Рубин (Andy Rubin), Рич Майнер (Rich Miner), Ник Сиэрс (Nick Sears) и Крис Уайт (Chris White).



Рис. 1.3. Хронология развития Android

В конце 2007 года группа лидирующих компаний, работающих в области мобильной связи, сплотилась вокруг платформы Android, сформировав Open Handset Alliance (http://www.openhandsetalliance.com). Некоторыми из наиболее известных членов альянса являются:

- Sprint Nextel;
- T-Mobile;
- Motorola;
- Samsung;
- Sony Ericsson;
- Toshiba;
- Vodafone;
- Google;
- Intel;
- Texas Instruments.

Одной из целей альянса является ускоренное внедрение инноваций и улучшение отклика на нужды потребителя — и первым значительным результатом работы альянса стала платформа Android. Она была разработана для удовлетворения потребностей операторов мобильной связи, производителей мобильных устройств и разработчиков программ. Члены альянса приняли решение предоставить значительный объем интеллектуальной собственности по свободной лицензии Apache, версия 2.0

ПРИМЕЧАНИЕ

Производители мобильных устройств не должны платить никаких лицензионных взносов за установку Android на свои телефоны или другие устройства.

Сначала инструментарий Android SDK был выпущен в ноябре 2007 года как предварительный вариант. В сентябре 2008 компания T-Mobile анонсировала выход T-Mobile G1, первого смартфона, работающего на платформе Android. Всего через несколько дней Google анонсировала выход Android SDK Release Candidate 1.0. В октябре 2008 года Google открыла доступ к исходному коду платформы Android в соответствии со свободной лицензией Apache.

Когда Android вышла в свет, одной из основных целей для ее архитектуры было обеспечить возможность взаимодействия приложений и использования компонентов одних приложений другими. Такое повторное применение касалось не только служб, но и данных, а также пользовательского интерфейса (UI). В результате платформа Android включила в себя некоторые архитектурные характеристики, которые позволили воплотить такую открытость в реальности. Некоторые из этих характеристик будут подробно рассмотрены в главе 3.

Android быстро привлекла к себе внимание, так как в ней имелись возможности, позволявшие в полной мере использовать модель облачных вычислений, которые применяются на веб-ресурсах, и этот опыт был усовершенствован при помощи локальных хранилищ данных, расположенных на самом мобильном устройстве. Поддержка реляционной базы данных на мобильных устройствах с Android также посодействовала быстрому признанию этой операционной системы.

В начале 2008 года Google выпустила мобильное устройство под названием Android Dev Phone 1, на котором можно было использовать программы Android, находясь вне зоны действия какой-либо мобильной сети. Это устройство (стоившее около $400) должно было позволить разработчикам проводить эксперименты с настоящими устройствами, на которых Android можно было использовать, не заключая никаких договоров. Примерно в то же время Google выпустила исправленную версию ОС под номером 1.1, которая была полностью основана на версии 1.0. В версиях 1.0 и 1.1 Android не поддерживала виртуальную клавиатуру, требовалось устройство, на котором были бы настоящие клавиши. Эта проблема была устранена в апреле 2009 года, когда вышел комплект SDK 1.5, в котором появились и другие возможности, например продвинутые функции записи медиаданных, виджеты и живые каталоги (живые каталоги будут рассмотрены в главе 12, а виджеты — в главе 13).

В сентябре 2009 вышла версия Android 1.6, а через месяц за ней последовал Android 2.0, поддержавший бурные продажи устройств Android в рождественский

сезон 2009 года. В этом релизе появились улучшенные поисковые возможности и функция преобразования текста в речь. (О синтезаторах речи мы поговорим в главе 15, а о поиске в Android — в главе 14.) В этом релизе появилась также поддержка управления жестами и функция мультитач. Эти вопросы рассмотрены в главе 16.

Android 2.0 поддерживает HTML 5, благодаря чему в нем появляются интересные функции. Новые возможности программирования описаны в главе 17, где мы рассмотрим Titanium Mobile. Каждый день появляются все новые приложения для Android, а также независимые онлайновые хранилища приложений. Эти хранилища приложений, а также управляемый Google онлайн-сервис Android Market, описаны в главе 18. В главе 19 мы проанализируем, насколько сильны позиции Android в сфере мобильных устройств.

Глубокий анализ Dalvik VM

В ходе работы с Android специалисты Google много внимания уделили возможностям оптимизации дизайна маломощных мобильных устройств. Мобильные устройства отстают от локальных компьютеров в области памяти и скорости обработки информации на восемь-десять лет. Вычислительные возможности мобильных устройств также ограниченны; общая оперативная память мобильного устройства может составлять всего 64 Мбайт, а пространство, на котором можно разместить приложения, может быть и того меньше — около 20 Мбайт.

В результате эксплуатационные требования к мобильным устройствам ужесточаются, заставляя разработчиков оптимизировать все, что только можно. Если просмотреть список пакетов для Android, становится понятно, что все они являются полнофункциональными, с широкими возможностями. По данным Google, эти системные библиотеки используют от 10 до 20 Мбайт памяти, даже при применении оптимизированной виртуальной машины Java.

По этим причинам Google пришлось во многих отношениях переработать стандартную версию JVM. (Основным специалистом Google, отвечающим за новую версию JVM является Дэн Борнштейн (Dan Bornstein), написавший виртуальную машину Dalvik VM; Дальвик — это небольшой город на севере Исландии.) Во-первых, Dalvik VM использует сгенерированные Java файлы классов и комбинирует их в один или несколько исполняемых файлов Dalvik (DEX). Машина повторно использует повторяющуюся информацию из нескольких файлов классов, эффективно снижая потребность в пространстве и занимает (без архивации) вполовину меньше места, чем обычный файл JAR. Например, файл DEX мобильного браузера, используемого в Android, имеет размер около 200 Кбайт, а аналогичный файл JAR без архивации занимает около 500 Кбайт. Файл DEX программы-будильника занимает 50 Кбайт, а версия JAR — без малого в два раза больше.

Во-вторых, Google тонко настроила в Dalvik VM процесс сборки мусора (garbage collection), но в ранних версиях было решено обойтись без динамического компилятора (JIT). Насколько можно судить, в коде версии 2.0 содержатся все необходимые источники для компилятора JIT, но в окончательной версии эти источники неактивированы. Ожидается, что они появятся в следующих релизах. Такой выбор

оправдан, так как многие корневые библиотеки Android, в том числе графические, написаны на C и C++. Например, графические интерфейсы прикладного программирования для Java являются тонкими классами-обертками, в которых заключен нативный код, использующий собственный интерфейс Java (Java Native Interface). Подобным образом в Android предоставляется оптимизированная нативная библиотека на базе C для доступа к базе данных SQLite, но эта библиотека инкапсулирована в высокоуровневом интерфейсе прикладного программирования (API) для Java. Поскольку большая часть корневого кода написана на C и C++, специалисты Google сочли, что компилирование JIT не очень повлияет на работу системы.

Наконец, в Dalvik VM используется иной вид генерирования ассемблерного кода, в ходе которого в качестве основных элементов хранения данных используются реестры, а не стек. Таким образом, Google рассчитывает сократить количество команд на 30 %. Следует отметить, что конечный исполняемый код в Android, получаемый после обработки Dalvik VM, основан не на байт-коде Java, а на файлах DEX. Это означает, что вы не сможете непосредственно выполнять байт-код Java; вам придется запустить файлы классов Java, а затем преобразовать их в готовые связываемые файлы DEX.

Такое настойчивое стремление оптимизировать эксплуатационные характеристики наблюдается во всех частях инструментария для разработки в Android. В частности, в Android SDK для определения пользовательских интерфейсов широко применяется XML. Но весь этот XML скомпилирован в двоичные файлы, которые затем становятся резидентными файлами мобильных устройств. В Android предусмотрены специальные механизмы для использования XML-данных. Поскольку мы рассматриваем вопросы дизайна Android, нам необходимо ответить на вопрос: чем похожи и чем различаются Android и платформа Java Micro Edition (Java ME)?

Сравнение Android и Java ME

Как мы уже увидели, в Android используется комплексный, целенаправленный и сфокусированный подход к созданию мобильной платформы, а для этого недостаточно обычных решений, основанных на JVM. В Android все, что вам нужно, — операционная система, драйверы устройств, библиотеки ядра, собственный интерфейс Java, оптимизированная версия Dalvik VM и среда разработки Java — находится в одном пакете. Разработчик может быть уверен, что при разработке нового приложения все основные библиотеки будут на мобильном устройстве.

Такой комплексный подход отличается от других решений, используемых в программировании для мобильных устройств, например от Java ME. Кратко рассмотрим Java ME, а потом сравним оба подхода. На рис. 1.4 показана доступность Java для различных конфигураций вычислительных машин. Стандартная версия платформы Java (Java SE) подходит для персональных компьютеров и рабочих станций. Корпоративная версия платформы Java (Java EE) разработана для серверов.

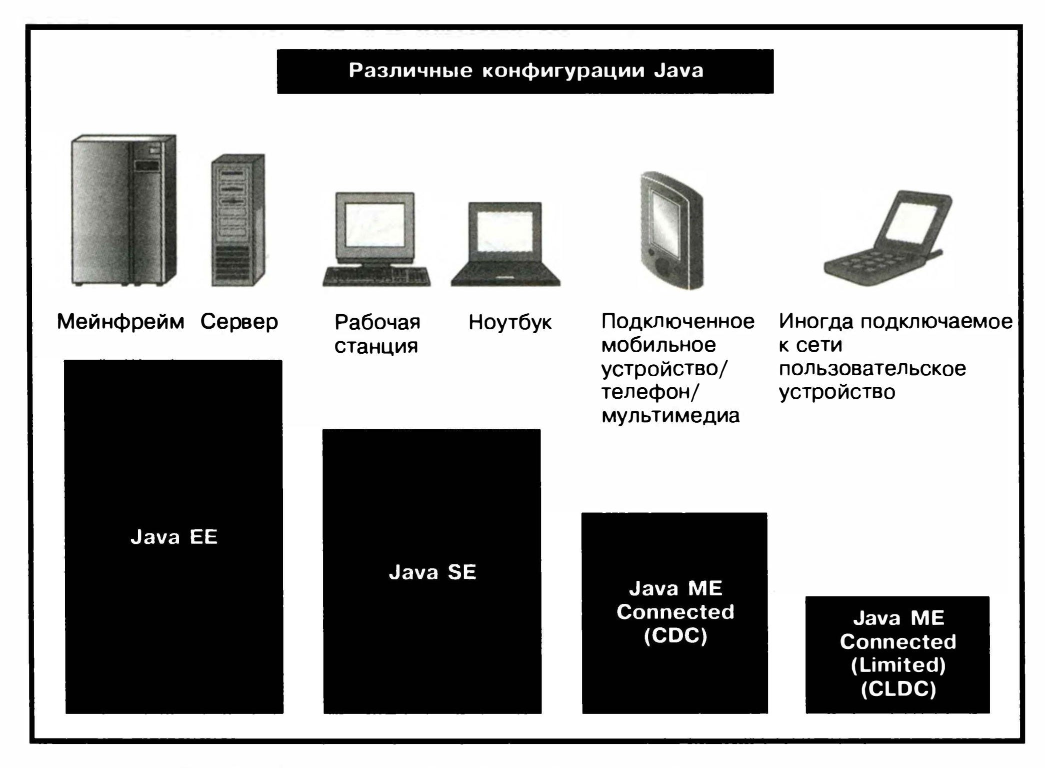


Рис. 1.4. Конфигурация Java для различных вычислительных машин

Микроверсия платформы Java (Java ME) является сокращенной и предназначена для небольших устройств. Java ME доступна в виде двух вариантов конфигурации. Первый вариант называется «конфигурация коммуникационных устройств» (Connected Device Configuration, CDC). Java ME для CDC содержит упрощенную версию Java SE — с меньшим количеством пакетов, с меньшим количеством классов в них, и даже с меньшим количеством методов и полей в этих классах. Для оборудования и устройств, имеющих дополнительные ограничения, в Java применяется конфигурация для устройств с ограниченными ресурсами (Connected Limited Device Configuration, CLDC). Существующие интерфейсы прикладного программирования для различных конфигураций Java сравниваются на рис. 1.5.

Любые дополнительные пакеты, устанавливаемые «поверх» основного интерфейса прикладного программирования с конфигурацией CDC или CLDC, считаются «профилями», которые стандартизируются процессом JSR (запрос на спецификацию Java). Каждый заданный профиль предоставляет разработчику дополнительный набор интерфейсов прикладного программирования.

ВНИМАНИЕ

И CDLC, и CDC могут поддерживать некоторые интерфейсы прикладного программирования Java, не входящие в Java SE, а их классы могут не начинаться с пространства имен *java. Следовательно, если у вас есть программа на Java, работающая на локальном компьютере, она может не сработать на устройствах, на которых поддерживается только микроверсия.

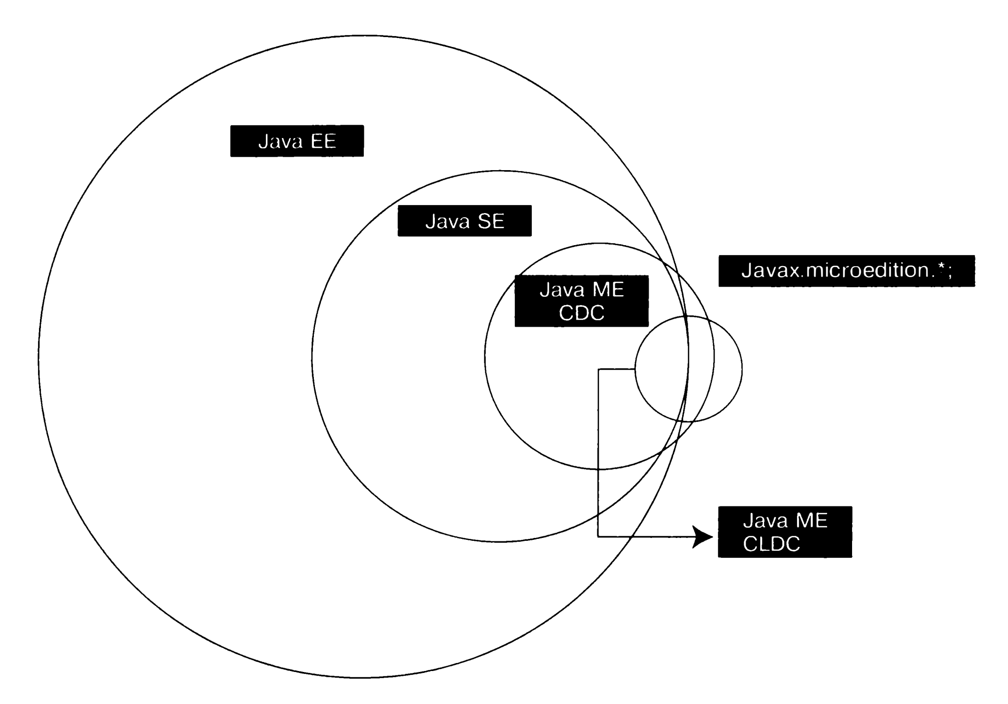


Рис. 1.5. Доступность интерфейсов прикладного программирования в Java

Платформа CLDC Java базируется на специализированной и сильно уменьшенной версии JVM, называемой «виртуальная машина K» (K Virtual Machine, KVM), которая способна работать на устройствах с памятью всего 128 Кбайт. В CLDC под MIDP 2.0 (профиль для мобильного устройства с информационными функциями) могут использоваться дополнительные интерфейсы прикладного программирования. Такой интерфейс содержит ряд пакетов для `javax.microedition.*`. К числу основных пакетов относятся мидлеты (простые приложения), пакет для пользовательских интерфейсов, называемый LCDUI, игровые и медиапакеты.

Интерфейсы прикладного программирования для конфигурации CDC содержат `java.awt` API и `java.net` API и некоторые API для реализации функций безопасности, дополнительно к тем интерфейсам, которые содержатся в CLDC. Дополнительные профили, устанавливаемые поверх CDC, открывают разработчикам приложений доступ к прикладному интерфейсу программирования `javax.microedition.xlet` (xlet — это приложение в конфигурации CDC). Поверх конфигурации CDC может находиться около десяти опциональных пакетов, которые вы можете использовать, в том числе Bluetooth, Media API, OpenGL для встроенных систем (OpenGL ES), Java API для обработки XML (JAXP), JAXP-RPC, Java 2D, Swing, интерфейс вызова удаленных методов Java (Java RMI), взаимодействие Java и баз данных (JDBC) и интерфейс прикладного программирования Java. Вообще, спецификация Java ME содержит более 20 JSR. Кроме того, ожидается, что при написании Java-программ для мобильных устройств JavaFX будет играть возрастающую роль (**http://javafx.com**).

ПРИМЕЧАНИЕ

JavaFX — это новый пользовательский интерфейс, разработка Sun, призванный радикально усовершенствовать апплетоподобные функции в браузерах. В нем используется декларативное UI-программирование, такая модель более удобна для разработчиков.

Теперь, когда мы рассмотрели основы Java ME, давайте сравним ее с Android.

- *Варианты конфигурации с несколькими устройствами*. В Java ME различаются два класса микроустройств, для каждого из которых предлагаются стандартные и особые решения. Android, в свою очередь, использует только одну модель. Она не будет работать с низкоуровневыми устройствами, если (или пока) их конфигурация не будет усовершенствована.
- *Понятность*. Поскольку Android ориентирована на работу с конкретным устройством, она понятнее, чем Java ME. В Java ME используется несколько вариантов пользовательских интерфейсов для каждой конфигурации, в зависимости от того, какие функции поддерживает устройство: мидлеты, икслеты (xlets), AWT и Swing. Отслеживать JSR для каждой спецификации Java ME сложнее. Они дольше разрабатываются, и найти определенные их версии может быть непросто.
- *Быстрота реагирования*. Ожидается, что Dalvik VM будет оптимизирована лучше и будет иметь более высокую скорость отклика, чем стандартная виртуальная машина JVM, используемая на устройстве с аналогичной конфигурацией. Можно сравнить Dalvik VM и KVM, но KVM предназначена для работы с менее сложными устройствами, располагающими значительно более скромными объемами памяти.
- *Совместимость с Java*. Поскольку Android работает с Dalvik VM, в ней используется не байт-код Java, а байт-код в файлах DEX. Это не очень большая проблема, если Java скомпилирована в виде стандартных файлов классов Java. Только непосредственная интерпретация байт-кода Java будет невозможна.
- *Широта внедрения*. Java ME широко поддерживается в мобильных устройствах, так как на ее основе работает большинство мобильных телефонов. Но Android отличается однородностью, дешевизной и простотой разработки программ — поэтому разработчиков Java могут заинтересовать и написание программ для Android.
- *Поддержка Java SE*. Если сравнить поддержку Java SE в CDC и в Android, то в Android такая поддержка реализована чуть более полно, если не считать AWT и Swing. Как уже упоминалось выше, в Android используется собственный подход к работе с пользовательскими интерфейсами. На самом деле декларативные пользовательские интерфейсы Android напоминают более продвинутые UI-платформы, такие как Microsoft Silverlight и JavaFX от Sun.

Программный стек Android

Итак, мы рассмотрели историю Android и ее компоненты, предназначенные для оптимизации работы, в том числе Dalvik VM. Кроме того, мы кратко рассказали

о программном стеке Java. В этом разделе мы опишем Android с точки зрения разработки программ в ней. Давайте начнем с рис. 1.6.

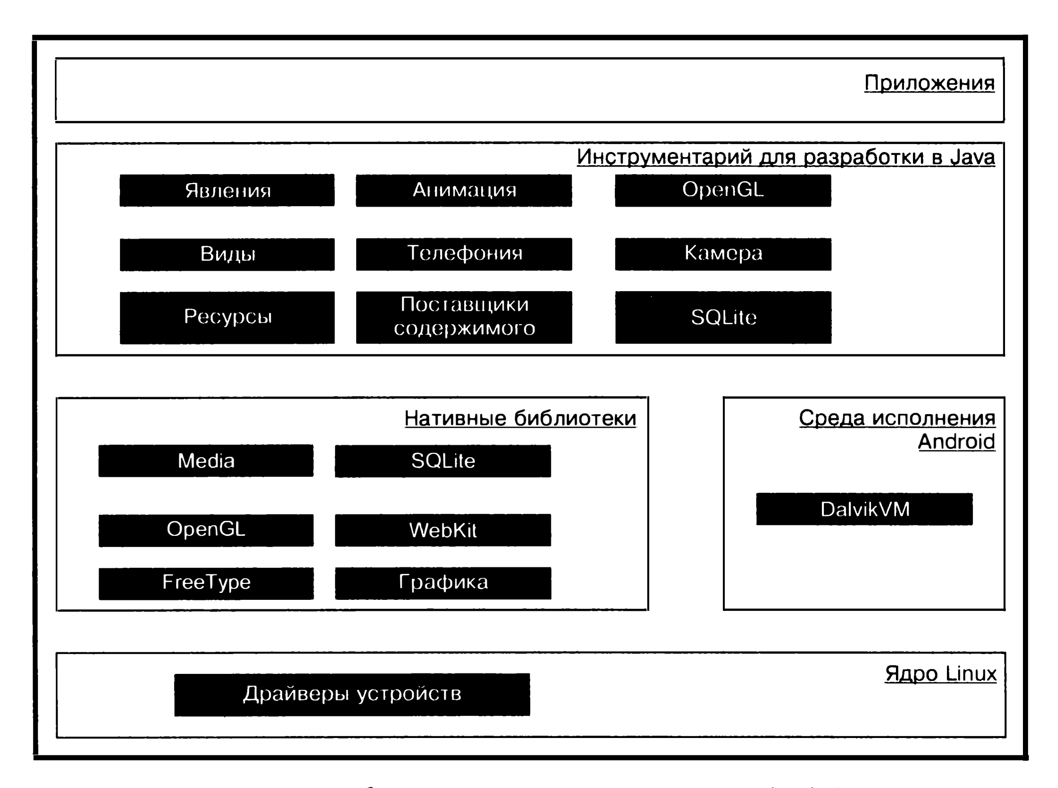


Рис. 1.6. Подробное описание программного стека Android SDK

Центром платформы Android является ядро Linux версии 2.6.29, отвечающее за драйверы устройств, доступ к ресурсам, управление энергопотреблением и решение других задач ОС. В такой сборке имеются драйверы устройств для работы с дисплеем, камерой, клавиатурой, Wi-Fi, флеш-памятью, аудио и для обеспечения связи между процессами (Inter-Process Communication, IPC). Хотя в системе и используется ядро Linux, подавляющее большинство приложений на устройствах Android (например, на T-Mobile G1 или Motorola Droid) разработаны на Java и работают при помощи Dalvik VM.

На следующем уровне, выше ядра, находится ряд библиотек C/C++, в частности OpenGL, WebKit, FreeType, Secure Sockets Layer (SSL), библиотека времени выполнения C (libc), SQLite и Media. Системная библиотека C, основанная на Berkeley Software Distribution (BSD), настроена для работы со встроенными устройствами, работающими под Linux (при этом ее размер уменьшен по сравнению с первоначальным примерно в два раза). Медиабиблиотеки работают на основе PacketVideo OpenCORE (http://www.packetvideo.com/). Эти библиотеки отвечают за запись и воспроизведение аудио- и видеоформатов. Библиотека, называемая Surface Manager, контролирует доступ к системе отображения данных и поддерживает 2D и 3D.

Библиотека WebKit отвечает за поддержку браузеров; именно эта библиотека поддерживает Google Chrome и Apple Safari. Библиотека FreeType поддерживает шрифты. SQLite (http://www.sqlite.org/) — это реляционная база данных, которая находится на самом устройстве. Кроме того, SQLite — это независимая разработка с открытым кодом, она не связана непосредственно с Android. Можно использовать инструменты, предназначенные для SQLite, и при работе с базами данных Android.

Большая часть приложений из этого набора обращается к указанным корневым библиотекам через Dalvik VM, выполняющую на платформе Android роль шлюза. Как уже говорилось в предыдущих разделах, Dalvik оптимизирована для одновременного использования нескольких экземпляров VM. Когда приложения Java обращаются к этим корневым библиотекам, каждое приложение работает с собственным экземпляром виртуальной машины.

В основных библиотеках прикладного интерфейса программирования на Java для Android содержатся функции для телефонии, работы с ресурсами, местоположением, пользовательскими интерфейсами, поставщиками содержимого (данными), а также диспетчеры пакетов (отвечающие за установку, безопасность и т. д.). Программисты разрабатывают приложения для конечных пользователей на основе данного прикладного интерфейса программирования Java. Примерами таких приложений являются Home, Contacts, Phone, Browser и т. д.

В Android также поддерживается пользовательская библиотека Google Skia, предназначенная для работы с 2D-графикой. Эта библиотека написана на C и C++. На Skia основан браузер Google Chrome. Однако прикладные интерфейсы для программирования 3D в Android работают на основе разновидности OpenGL ES от группы Khronos (http://www.khronos.org). В OpenGL ES содержатся сокращенные версии OpenGL, оптимизированные для работы со встроенными системами.

Что касается медиа, на платформе Android поддерживается большинство распространенных форматов аудио, видео и изображений. В области беспроводной связи Android располагает специальными API для поддержки Bluetooth, EDGE, 3G, Wi-Fi и глобальной системы мобильной связи (GSM), зависимыми от оборудования.

Разработка готовых пользовательских приложений при помощи Android SDK

В данном разделе мы познакомимся с высокоуровневым прикладным интерфейсом программирования Java для Android, при помощи которого мы впоследствии будем создавать приложения для конечных пользователей Android. Мы кратко обсудим эмулятор Android, фундаментальные компоненты Android, программирование пользовательских интерфейсов, сервисы, медиа, телефонию, анимацию и OpenGL. Кроме того, мы рассмотрим несколько фрагментов кода.

Эмулятор Android

Комплект разработки программ (SDK) для Android поставляется вместе с плагином Eclipse, который называется инструментарием для разработки в Android (Android Development Tools, ADT). Этот инструмент интегрированной среды разработки

(IDE) используется для создания, отладки и тестирования приложений на Java (подробнее ADT будет рассмотрен в главе 2). Android SDK можно использовать и без ADT; вместо этого инструментария можно применять средства командной строки. Эмулятор поддерживается при использовании обоих подходов, и с его помощью вы можете запускать, исправлять и тестировать свои приложения. 90 % разработки приложения можно завершить, вообще не пользуясь реальным устройством. Полнофункциональный эмулятор Android воспроизводит большинство характеристик устройств. К числу тех функций, которые нельзя имитировать в эмуляторе, относятся USB-соединения, работа камеры и видеосъемка, имитация работы наушников, батарей и технология Bluetooth.

Эмулятор Android работает на базе свободно распространяемой технологии «имитации процессора», называемой QEMU (http://bellard.org/qemu/), которую разработал Фабрис Беллар (Fabrice Bellard). Та же технология позволяет эмулировать одну операционную систему в другой, независимо от того, какой процессор применяется. QEMU обеспечивает эмуляцию на уровне процессора.

При работе эмулятора Android имитируется процессор, функционирующий на базе ARM (Advanced RISC Machine, усовершенствованная RISC-машина). ARM — это 32-битная архитектура микропроцессоров, основанная на RISC (Reduced Instruction Set Computer, компьютер с сокращенным набором команд), в которой благодаря уменьшению числа команд достигается простой дизайн и увеличение скорости работы. Эмулятор задействует на таком имитируемом процессоре версию Linux, используемую в Android.

ПРИМЕЧАНИЕ

Многие высокотехнологичные и научные рабочие станции HP и Sun работают на базе усовершенствованных процессоров RISC.

ARM широко используется в мобильных устройствах и во встроенных электронных приборах, где важно обходиться небольшим количеством энергии. Многие имеющиеся на рынке мобильные устройства имеют процессоры с такой архитектурой. Например, Apple Newton основан на процессоре ARM6. Устройства iPod, Nintendo DS и Game Boy Advance работают на архитектуре ARM версии 4, в которой используется около 30 000 транзисторов. Классический Pentium содержит 3,2 млн транзисторов.

Более подробная документация эмулятора, используемого в Android SDK, приведена по адресу http://developer.android.com/guide/developing/tools/emulator.html.

Пользовательский интерфейс Android

В Android используется UI-фреймворк, сравнимый с другими полнофункциональными UI-фреймворками, применяемыми на локальных компьютерах. На самом деле он более современный и асинхронный по природе. По существу, UI-фреймворк Android относится уже к четвертому поколению, если считать первым поколением традиционный прикладной интерфейс программирования Microsoft Windows, основанный на C, а MFC (Microsoft Foundation Classes, библиотека базовых классов Microsoft на основе C++) — вторым. В таком случае UI-фреймворк Swing, основанный на Java, будет третьим поколением, так как предлагаемые в нем воз-

можности дизайна значительно превосходят по гибкости MFC. Android UI, JavaFX, Microsoft Silverlight и язык пользовательских интерфейсов Mozilla XML (XUL) относятся к новому типу UI-фреймворков четвертого поколения, в котором UI является декларативным и поддерживает независимую темизацию.

ПРИМЕЧАНИЕ

При программировании в Android используется современная парадигма создания пользовательского интерфейса, хотя программа и пишется для мобильного устройства.

При программировании в пользовательском интерфейсе Android применяется объявление интерфейса в файлах XML. Затем эти определения представления (view definitions) XML загружаются в приложение с пользовательским интерфейсом как окна. Даже меню вашего приложения загружаются из файлов XML. Экраны (окна) Android часто называются ***явлениями (activities)***, которые включают в себя несколько видов, нужных пользователю, чтобы выполнить логический элемент процесса. ***Виды (views)*** являются основными элементами, из которых в Android состоит пользовательский интерфейс. Виды можно объединять в группы ***(view groups)***. Для внутренней организации видов используются давно известные в программировании концепции холст (canvas), рисование (painting) и взаимодействие пользователя с системой (user interaction).

Такие составные представления, в которые входят виды и группы видов, работают на базе специального логического заменяемого компонента пользовательского интерфейса Android.

Одной из ключевых концепций фреймворка Android является управление жизненным циклом (lifecycle) окон явлений (activity windows). В системе применяются протоколы, поэтому Android может управлять ситуацией по мере того, как пользователи скрывают, восстанавливают, останавливают и закрывают окна явлений. Эти базовые идеи станут понятны вам после прочтения главы 2, в которой мы рассмотрим подготовку среды для разработки в Android.

Базовые компоненты Android

Фреймворк пользовательского интерфейса Android вместе с другими компонентами Android базируется на новой сущности, называемой ***намерением (intent)***. Намерение — это сложное явление, в котором сочетаются такие идеи, как сообщения, выводимые в окнах (windowing messages), действия (actions), модели типа «публикация и подписка» (publish-and-subscribe), межпроцессный обмен информацией и реестры приложений. Ниже приведен пример использования класса `Intent` для активации или запуска веб-браузера:

```
public static void invokeWebBrowser(Activity activity)
{
   Intent intent = new Intent(Intent.ACTION_VIEW);
   intent.setData(Uri.parse("http://www.google.com"));
   activity.startActivity(intent);
}
```

В данном примере, используя намерение, мы говорим Android открыть окно, подходящее для отображения контента веб-сайта. В зависимости от того, какие

браузеры установлены на мобильном устройстве, Android выберет для отображения сайта наиболее подходящий. Более подробно намерения будут рассмотрены в главе 3.

Кроме того, в Android широко поддерживаются *ресурсы (resources)*, к которым относятся хорошо знакомые вам строки (strings) и растровые изображения (bitmaps), а также некоторые менее известные элементы, в частности основанные на XML определения представления. Использование ресурсов в данном фреймворке осуществляется новым способом, благодаря которому работа с ресурсами становится более простой, понятной и удобной. Ниже приведен пример, в котором автоматически генерируются ID ресурсов, определенных в файлах XML:

```
public final class R {
   public static final class attr { }
   public static final class drawable {
      public static final int myanimation=0x7f020001;
      public static final int numbers19=0x7f02000e;
   }
   public static final class id {
      public static final int textViewId1=0x7f080003;
   }
   public static final class layout {
      public static final int frame_animations_layout=0x7f030001;
      public static final int main=0x7f030002;
   }
   public static final class string {
      public static final int hello=0x7f070000;
   }
}
```

В данных классах автоматически сгенерированные ID соответствуют либо элементу XML-файла, либо целому такому файлу. Если вам понадобится использовать такие определения XML, вместо них будут применяться данные ID. Такая опосредованность очень помогает при локализации (в главе 3 более подробно рассмотрены файл `R.java` и ресурсы).

Еще одна новаторская концепция в Android — это ***поставщик содержимого (content provider)***. Под поставщиком содержимого понимается абстракция источника данных, который можно представить как эмиттер и потребитель служб REST (Representational State Transfer, передача состояния представления). Лежащая в основе этой абстракции база данных SQLite делает данное свойство поставщиков содержимого мощным инструментом для разработки приложений (в главе 3 будет рассмотрено, как намерения, ресурсы и поставщики содержимого способствуют открытости платформы Android).

Продвинутые концепции пользовательского интерфейса

Мы уже отмечали, что XML играет важнейшую роль в описании пользовательских интерфейсов Android. Теперь на примере рассмотрим, как XML выполня-

ет эту задачу с простым шаблоном, в котором осуществляется представление текста:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android>
<TextView android:id="@+id/textViewId"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
</LinearLayout>
```

Мы будем применять ID, сгенерированный для данного XML-файла, чтобы загрузить этот шаблон в окно действия. (Более подробно этот процесс будет рассмотрен в главе 4.) В Android также широко поддерживаются меню, от стандартных до контекстных. Вам будет очень удобно работать с меню в Android, поскольку они также загружаются как XML-файлы и поскольку ID ресурсов для этих меню генерируются автоматически. Ниже показано, как объявлять меню в XML-файле:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <!—В этой группе используется категория, заданная по умолчанию. -->
   <group android:id="@+id/menuGroup_Main">
      <item android:id="@+id/menu_clear"
         android:orderInCategory="10"
         android:title="clear" />
      <item android:id="@+id/menu_show_browser"
         android:orderInCategory="5"
         android:title="show browser" />
   </group>
</menu>
```

Хотя в Android и поддерживаются диалоговые окна, все они являются асинхронными. Работа с такими асинхронными диалоговыми окнами представляет особую сложность для разработчиков, привыкших к работе с синхронными модальными диалоговыми окнами, которые используются в некоторых оконных фреймворках. Мы более подробно рассмотрим меню и диалоговые окна в главе 5, где также обсудим некоторые механизмы, используемые при работе с протоколами асинхронных диалоговых окон.

В Android также поддерживается анимация — эта функция входит в состав стека пользовательского интерфейса, который базируется на видах и отрисовываемых объектах. В Android используется анимация двух видов: с построением промежуточных кадров (tweening animation) и покадровая (frame-by-frame animation). *Промежуточными* в анимации называются такие рисунки, которые отображаются *между* основными. На компьютере эти рисунки создаются методом изменения средних значений через определенные промежутки времени и перерисовывания фона. Покадровая анимация состоит из серий кадров, которые прорисовываются один за другим через регулярные временные интервалы. В Android применяются оба варианта анимации, при этом используются функции обратного вызова анимации, интерполяторы и матрицы преобразований. Кроме того, в Android можно

определять такие виды анимации в файле ресурсов XML. В следующем примере показаны серии пронумерованных изображений, которые воспроизводятся при покадровой анимации:

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
        android:oneshot="false">
    <item android:drawable="@drawable/numbers11" android:duration="50" />
    ......
    <item android:drawable="@drawable/numbers19" android:duration="50" />
</animation-list>
```

Графические библиотеки, на которых основан такой процесс, поддерживают стандартные матрицы преобразований, позволяя масштабировать, перемещать и вращать рисунки. Объект Camera, присутствующий в графической библиотеке, обеспечивает поддержку глубины и проекции, благодаря чему на двухмерном интерфейсе удается имитировать трехмерные эффекты (более подробно анимация будет рассмотрена в главе 6).

В Android поддерживается и трехмерная графика. Это происходит благодаря тому, что в систему внедрен стандарт OpenGL ES 1.0. Подобно OpenGL, он является плоским API на базе языка C. Поскольку интерфейс прикладного программирования Android SDK базируется на Java, для доступа к OpenGL в нем следует использовать Java-связывание. В Java ME такое связывание для OpenGL ES уже определено при помощи запроса на спецификацию Java (JSR) 239, и в Android для OpenGL ES используется такой же вид Java-связывания. Если вы не знакомы с программированием OpenGL, то вам придется его выучить, причем достаточно глубоко. В этой книге мы рассмотрели основы данной проблемы, поэтому, освоив главу 10 данной книги, вы сможете начать программировать в OpenGL для Android.

В Android используются новые подходы, связанные с идеей *информации на кончиках пальцев* (information at your fingertips), доступ к которой осуществляется через домашнюю страницу. Первая из таких идей называется *живые каталоги* (live folders). При помощи живых каталогов можно опубликовать коллекцию элементов в виде папки, расположенной на домашней странице. Содержимое этой коллекции меняется по мере того, как изменяются лежащие в ее основе данные. Новые данные могут поступать на устройство либо со съемных носителей, либо из Интернета (мы рассмотрим живые каталоги в главе 12).

Вторая идея, связанная с домашней страницей, — это *домашний виджет* (home screen widget). Домашние виджеты используются для отрисовывания информации на домашней странице с применением виджета пользовательского интерфейса. Данная информация может изменяться через регулярные временные интервалы. Примером такой информации может быть несколько электронных сообщений, которые сохранены на устройстве. Мы рассмотрим домашние виджеты в главе 13.

Интегрированный поиск Android (Integrated Android Search) — это третья идея, связанная с использованием домашней страницы. Такой вид поиска позволяет искать информацию как на самом устройстве, так и в Интернете. Данная функция Android не ограничивается одним только поиском и позволяет давать команды при помощи элемента управления поиском. Поиск в Android рассмотрен в главе 14.

Кроме того, в Android поддерживаются так называемые жесты, то есть интерпретация движений пальцев пользователя, работающего с устройством. Android позволяет записывать любые последовательности движений пальцев по экрану и сохранять их как жесты. Затем такие жесты могут использоваться приложениями для обозначения конкретных действий. Сенсорные экраны и работа с жестами рассматриваются в главе 16.

Кроме инструментария Android SDK, существуют другие самостоятельные инновации, которые делают процесс разработки интересным и несложным. Некоторые примеры таких явлений — XML/VM, PhoneGap и Titanium. Titanium позволяет использовать технологии HTML при программировании для основанного на WebKit браузера Android. Такой подход к разработке пользовательских интерфейсов очень пластичен и интересен, более подробно мы рассмотрим его в главе 17.

Служебные компоненты Android

Важнейшим аспектом платформы Android является безопасность. В Android безопасности внимание уделяется на всех этапах жизненного цикла приложения — от соблюдения политик в ходе разработки до проверок граничных условий в ходе выполнения программы. Другим интересным компонентом Android SDK являются сервисы, основанные на местоположении. Данный раздел SDK предоставляет разработчикам приложений интерфейсы прикладного программирования, предназначенные для работы с картами, а также для получения в реальном времени информации, связанной с местоположением устройства. Эти концепции будут рассмотрены в главе 7.

В главе 8 мы рассмотрим, как строятся и используются службы в Android, в частности HTTP-службы. Кроме того, в этой главе мы изучим межпроцессный обмен информацией (коммуникация между приложениями, расположенными на одном и том же устройстве).

Ниже приведен пример запроса `HttpPost` в Android:

```
InputStream is = this.getAssets().open("data.xml");
HttpClient httpClient = new DefaultHttpClient();
HttpPost postRequest = new HttpPost("http://192.178.10.131/WS2/Upload.aspx");

byte[] data = IOUtils.toByteArray(is);

InputStreamBody isb = new InputStreamBody(
            new ByteArrayInputStream(data),"zagruzhennyjFile");
StringBody sb1 = new StringBody("TutNapisanText");
StringBody sb2 = new StringBody("TutTozheNapisanText ");

MultipartEntity multipartContent = new MultipartEntity();
multipartContent.addPart("zagruzhennyjFile", isb);
multipartContent.addPart("odin", sb1);
multipartContent.addPart("dva", sb2);

postRequest.setEntity(multipartContent);
HttpResponse res =httpClient.execute(postRequest);
res.getEntity().getContent().close();
```

Медийные компоненты Android и компоненты, связанные с телефонией

В Android имеются интерфейсы API, предназначенные для работы с аудио-, видео- и телефонными компонентами. Ниже приведен пример того, как воспроизвести аудиофайл из Интернета по URL:

```
private void playAudio(String url)throws Exception
{
   mediaPlayer = new MediaPlayer();
   mediaPlayer.setDataSource(internetUrl);
   mediaPlayer.prepare();
   mediaPlayer.start();
}
```

А вот как аудиофайл воспроизводится с локального устройства:

```
{
   // файл расположен в каталоге /res/raw и называется "music_file.mp3"
   mediaPlayer = MediaPlayer.create(this, R.raw.music_file);
   mediaPlayer.start();
}
```

Данные интерфейсы прикладного программирования будут подробно рассмотрены в главе 9. Кроме того, в этой главе мы поговорим о таких функциях API, используемых в телефонии, как:

- отправка и получение коротких сообщений (SMS);
- отслеживание SMS-сообщений;
- управление каталогами с SMS;
- установление и прием телефонных вызовов.

Ниже показан пример кода для отправки SMS-сообщения:

```
private void sendSmsMessage(String address,String message)throws Exception
{
   SmsManager smsMgr = SmsManager.getDefault();
   smsMgr.sendTextMessage(address, null, message, null, null);
}
```

До версии 1.5 в Android можно было записывать только аудио, а видео — нет. В версии 1.5 была обеспечена возможность записи как аудио, так и видео. Это было сделано при помощи MediaRecorder. В главе 9 также рассмотрен вопрос распознавания голоса и набор методов для реализации ввода (IMF), благодаря которому широкий спектр вводимой информации может быть интерпретирован как текст, когда вы записываете такие данные в элементы, предназначенные для работы с текстом. Ввод может осуществляться посредством клавиатуры, голоса, стилуса, мыши и т. д. Данный фреймворк первоначально разрабатывался как часть Java API 1.4; более подробно об этом можно прочитать на следующем сайте: http://java.sun.com/j2se/1.4.2/docs/guide/imf/overview.html.

В версиях Android 2.0 и выше применяется движок Pico для преобразования текста в речь (синтеза речи). Интерфейс, используемый в Android для преобразования текста в речь, очень прост, как и соответствующий код:

```
TextToSpeech mTTS;
….
mTTS.speak(sometextString, TextToSpeech.QUEUE_ADD);
…
mTTS.setOnUtteranceCompletedListener(this);
….
mTTS.stop();
….
mTTS.shutdown();
…
mTTS.synthesizeToFile(…)
```

Еще несколько методов, относящихся к этой сфере:

```
playSilence
setLanguage
setPitch
setSpeechRate
isSpeaking
```

Более подробно они будут рассмотрены в главе 15.

И наконец, важно отметить, что в Android все эти концепции объединены в приложение путем создания единого XML-файла, в котором определяется, каким будет пакет прикладных программ. Этот файл называется файлом описания (manifest file) (AndroidManifest.xml). Вот пример такого файла:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.ai.android.HelloWorld"
      android:versionCode="1"
      android:versionName="1.0.0">
   <application android:icon="@drawable/icon"
                 android:label="@string/app_name">
         <activity android:name=".HelloWorld"
                 android:label="@string/app_name">
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category
                  android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
         </activity>
   </application>
<-/manifest>
```

В файле описания Android содержатся определения действий, регистрируются поставщики содержимого и поставщики служб и обозначаются права доступа. Мы будем возвращаться к файлу описания на протяжении книги по мере развития тех или иных идей.

Пакеты Java для Android

Один из способов быстро составить представление о платформе Android — рассмотреть структуру пакетов Java. Поскольку Android отличается от стандартного дистрибутива SDK, важно знать, какие пакеты поддерживаются, а какие — нет. Ниже приводится краткое описание важных пакетов, входящих в состав Android SDK:

- *android.app* — реализует модель приложений для Android. Среди основных классов — `Application`, в котором описаны начальная и конечная семантика, а также ряд классов, относящихся к явлениям, элементы управления, диалоговые окна, окна с предупреждениями и уведомлениями;
- *android.bluetooth* — содержит классы для работы с технологией Bluetooth. К числу основных классов относятся `BluetoothAdapter`, `BluetoothDevice`, `BluetoothSocket`, `BluetoothServerSocket` и `BluetoothClass`. Класс `BluetoothAdapter` можно использовать для управления адаптером Bluetooth, установленным на локальном компьютере. Этот адаптер можно включать, отключать или запускать процесс обнаружения. Класс `BluetoothDevice` представляет собой дистанционное устройство Bluetooth, к которому вы можете подключиться. Для установки связи между устройствами используются два сокета Bluetooth. Класс `Bluetooth` представляет собой тип устройства Bluetooth, к которому вы подключаетесь;
- *android.content* — реализует концепции, связанные с поставщиками содержимого. Поставщик содержимого позволяет обобщать обмен данными и их хранение. Кроме того, в данном пакете реализуются основные идеи, касающиеся намерений и унифицированных идентификаторов ресурсов (URI) в Android;
- *android.content.pm* — обеспечивает работу классов, относящихся к диспетчеру пакетов. Он располагает информацией о правах доступа, установленных пакетах, установленных поставщиках, службах и компонентах, например таких, как действия, а также об установленных приложениях;
- *android.content.res* — обеспечивает доступ к файлам ресурсов, как структурированным, так и неструктурированным. Основными классами являются `AssetManager` (для неструктурированных ресурсов) и `Resources`;
- *android.database* — реализует идею реферативной базы данных. Основной интерфейс называется `Cursor`;
- *android.database.sqlite* — реализует концепции из пакета `android.database`, используя в качестве физической базы данных SQLite. Основными классами являются `SQLiteCursor`, `SQLiteDatabase`, `SQLiteQuery`, `SQLiteQueryBuilder` и `SQLiteStatement`. Правда, в основном вам придется работать с классами из абстрактного пакета `android.database`;
- *android.gesture* — в этом пакете располагаются все классы и интерфейсы, необходимые для работы с заданными пользователем жестами. Основными классами являются `Gesture`, `GestureLibrary`, `GestureOverlayView`, `GestureStore`, `GestureStroke`, `GesturePoint`. Класс `Gesture` является подборкой `GestureStrokes` и `GesturePoints`. Жесты собраны в библиотеке `GestureLibrary`. Библиотеки жестов сохраняются

в GestureStore. Имена жестов таковы, что система может идентифицировать их как действия;

- *android.graphics* — содержит классы Bitmap, Canvas, Camera, Color, Matrix, Movie, Paint, Path, Rasterizer, Shader, SweepGradient и TypeFace;
- *android.graphics.drawable* — предназначен для работы с протоколами рисования и фоновыми рисунками, обеспечивает анимационные эффекты при работе с отрисовываемыми объектами;
- *android.graphics.drawable.shapes* — обеспечивает работу с контурами, в том числе ArcShape, OvalShape, PathShape, RectShape и RoundRectShape;
- *android.hardware* — обеспечивает использование так называемых физических классов, предназначенных для работы с камерой. Класс Camera представляет собой обычное устройство-камеру, а класс android.graphics.Camera — графическую концепцию, не имеющую никакого отношения к физической реальной камере;
- *android.location* — содержит классы Address, GeoCoder, Location, LocationManager и LocationProvider. Класс Address представляет собой упрощенный язык XAL (Extensible Address Language, расширяемый язык адресов). GeoCoder позволяет узнать по адресу координаты объекта (широту и долготу) и наоборот. Location представляет информацию о широте и долготе;
- *android.media* — содержит классы MediaPlayer, MediaRecorder, Ringtone, AudioManager и FaceDetector. Класс MediaPlayer предназначен для работы с потоками (streaming) и поддерживает аудио и видео. Класс Ringtone используется для проигрывания коротких звуковых фрагментов, которые могут служить рингтонами или использоваться при уведомлениях. AudioManager отвечает за контроль громкости. FaceDetector можно применять для нахождения человеческих лиц на точечных (растровых) рисунках;
- *android.net* — реализует основные сетевые API на уровне сокетов. Основные классы включают Uri, ConnectivityManager, LocalSocket и LocalServerSocket. Здесь также следует отметить, что Android поддерживает HTTPS на уровне браузера и на уровне сети. Кроме того, Android поддерживает в браузере JavaScript;
- *android.net.wifi* — управляет соединяемостью по Wi-Fi. К основным классам относятся WifiManager и WifiConfiguration. Класс WifiManager отвечает за составление списка сконфигурированных сетей и за работу с активной в настоящее время сетью Wi-Fi;
- *android.opengl* — содержит вспомогательные классы, используемые при выполнении операций OpenGL ES. Основные классы OpenGL ES входят в состав другого набора пакетов, взятого из JSR 239. Это пакеты:
  - javax.microedition.khronos.opengles;
  - javax.microedition.khronos.egl;
  - javax.microedition.khronos.nio.

  Данные пакеты являются тонкими оболочками, охватывающими версию Khronos для OpenGL ES, которая написана на C и C++;

- *android.os* — здесь находятся службы операционной системы, доступ к которой осуществляется средствами языка Java. Некоторые важные классы — `BatteryManager`, `Binder`, `FileObserver`, `Handler`, `Looper` и `PowerManager`. Класс `Binder` обеспечивает обмен информацией между процессами. `FileObserver` ведет учет изменений, вносимых в файлы. Класс `Handler` используется для выполнения задач в рамках потока сообщений, а `Looper` запускает сам поток сообщений;
- *android.preference* — позволяет приложениям предоставлять пользователям возможность управления своими настройками для этого приложения в унифицированной форме. Основными классами являются `PreferenceActivity`, `PreferenceScreen` и различные классы, производные от `Preference`, например `CheckBoxPreference` и `SharedPreferences`;
- *android.provider* — включает в себя набор предварительно подготовленных поставщиков содержимого, относящихся к интерфейсу `android.content.ContentProvider`. Среди поставщиков содержимого — `Contacts`, `MediaStore`, `Browser` и `Settings`. В данном наборе интерфейсов и классов хранятся метаданные для описания базовых структур данных;
- *android.sax* — содержит эффективный набор простых API для XML (SAX), вспомогательных классов, предназначенных для синтаксического разбора. К основным классам относятся `Element`, `RootElement` и некоторые интерфейсы `ElementListener`;
- *android.speech* — содержит константы для работы с распознаванием речи. Этот пакет включен только в версии 1.6 и выше;
- *android.speech.tts* — обеспечивает поддержку преобразования текста в речь. Основной класс — `TextToSpeech`. Можно взять фрагмент текста и запросить экземпляр этого класса поставить текст в очередь для воспроизведения в виде речи. У вас появится доступ к нескольким обратным вызовам, которые позволят вам наблюдать за речью — например узнать, когда завершится воспроизведение. В Android используется механизм PICO TTS (Text to Speech, синтезатор речи) производства компании SVOX;
- *android.telephony* — содержит классы `CellLocation`, `PhoneNumberUtils` и `TelephonyManager`. Класс `TelephonyManager` позволяет определить место, откуда был сделан вызов, номер телефона, название оператора связи, тип сети, тип телефона и серийный номер модуля идентификации абонента (Subscriber Identity Module, SIM);
- *android.telephony.gsm* — позволяет собирать информацию об адресах ячеек на основании данных о местонахождении вышек сотовой связи, а также содержит классы, отвечающие за работу с сообщениями SMS. В названии этого пакета упоминается GSM, так как первоначально стандарты обмена короткими сообщениями (SMS) определялись Глобальной системой мобильной связи (Global System for Mobile Communication);
- *android.telephony.cdma* — обеспечивает поддержку телефонии стандарта CDMA;
- *android.text* — содержит классы для обработки текста;
- *android.text.method* — предоставляет классы для ввода текста в различные элементы управления;

- *android.text.style* — предоставляет различные методы оформления фрагмента текста;
- *android.utils* — содержит классы Log, DebugUtils, TimeUtils и Xml;
- *android.view* — содержит классы Menu, View, ViewGroup, а также некоторые процессы-слушатели и обратные вызовы;
- *android.view.animation* — обеспечивает поддержку анимации с построением промежуточных кадров. К основным классам относятся Animation, а также некоторые интерполяторы анимации и специфические анимационные классы, среди которых AlphaAnimation, ScaleAnimation, TranslationAnimation и RotationAnimation;
- *android.view.inputmethod* — реализует архитектуру фреймворка ввода-вывода. Этот пакет содержится только в версиях 1.5 и выше;
- *android.webkit* — содержит классы, относящиеся к веб-браузеру. Среди основных классов WebView, CacheManager и CookieManager;
- *android.widget* — содержит все классы элементов управления пользовательского интерфейса, которые в основном являются производными класса view. Основные виджеты — Button, Checkbox, Chronometer, AnalogClock, DatePicker, DigitalClock, EditText, ListView, FrameLayout, GridView, ImageButton, MediaController, ProgressBar, RadioButton, RadioGroup, RatingButton, Scroller, ScrollView, Spinner, TabWidget, TextView, TimePicker, VideoView и ZoomButton;
- *com.google.android.maps* — содержит классы MapView, MapController и MapActivity, необходимые для работы с картами Google.

Упомянутые выше пакеты очень важны при работе с Android. На основании этого списка вы можете составить представление о глубинном строении платформы Android.

ПРИМЕЧАНИЕ

В целом интерфейс прикладного программирования Android Java включает более 40 пакетов и более 700 классов.

Кроме того, в Android имеется ряд пакетов, находящихся в пространстве имен java.*. К их числу относятся awt.font, io, lang, lang.annotation, lang.ref, lang.reflect, math, net, nio, nio.channels, nio.channels.spi, nio.charset, security, security.acl, security.cert, security.interfaces, security.spec, sql, text, util, util.concurrent, util.concurrent.atomic, util.concurrent.locks, util.jar, util.logging, util.prefs, util.regex и util.zip. Android работает со следующими пакетами из пространства имен javax: crypto, crypto.spec, microedition.khronos.egl, microedition.khronos.opengles, net, net.ssl, security.auth, security.auth.callback, security.auth.login, security.auth.x500, security.cert, sql, xml и xmlparsers. Этим список пакетов не ограничивается: еще используется множество пакетов из org.apache.http.*, а также org.json, org.w3c.dom, org.xml.sax, org.xml.sax.ext, org.xml.sax.helpers, org.xmlpull.v1 и org.xmlpull.v1.sax2. Вместе все эти многочисленные пакеты составляют насыщенную вычислительную платформу, предназначенную для написания программ для мобильных устройств.

Использование преимуществ исходного кода Android

Документация по ранним версиям Android была неполной. Для заполнения пробелов можно воспользоваться исходным кодом Android.

Исходный дистрибутив Android опубликован по адресу http://source.android.com. Код был предоставлен на правах свободного ПО в октябре 2008 года (см. анонс по адресу http://source.android.com/posts/opensource). Одной из целей Open Handset Alliance ставит развитие Android как свободной и полностью настраиваемой мобильной платформы. В анонсе особо подчеркивается, что Android — это полноценная вычислительная платформа, приспособленная для решения любых задач. Благодаря использованию принципа свободного ПО в развитии платформы могут участвовать специалисты из открытых сообществ.

Как уже говорилось выше, Android — это платформа, а не одиночный проект. По адресу http://source.android.com/projects вы можете ознакомиться с тем, в каких сферах реализуются различные проекты для Android, а также узнать о количестве этих проектов.

Исходный код Android и всех его проектов управляется системой для контроля исходного кода Git (http://git.or.cz/). Это свободная система для управления исходным кодом, предназначенная для быстрой и удобной работы с большими и малыми проектами. Проекты, связанные с разработкой ядра Linux и Ruby on Rails, также используют Git для контроля версий. Полный список проектов Android, занесенных в репозиторий Git, находится по адресу http://android.git.kernel.org/.

Вы можете скачать любые из этих проектов, пользуясь специальными инструментами Git, — для описания каждого из них создан отдельный сайт. К числу основных проектов относятся Dalvik, frameworks/base (файл `android.jar`), ядро Linux и некоторые внешние библиотки, в частности библиотеки Apache HTTP (apache-http). Здесь же расположены основные приложения Android. К ним относятся: AlarmClock, Browser, Calculator, Calendar, Camera, Contacts, Email, GoogleSearch, HTML Viewer, IM, Launcher, Mms, Music, PackageInstaller, Phone, Settings, SoundRecorder, Stk, Sync, Updater и VoiceDialer.

Часть проектов Android связана с созданием поставщиков. *Проекты поставщиков* в Android напоминают базы данных — они заключают свои данные в оболочки из служб RESTful. К таким проектам относятся CalendarProvider, ContactsProvider, DownloadProvider, DrmProvider, GoogleContactsProvider, GoogleSubscribedFeedsProvider, ImProvider, MediaProvider, SettingsProvider, Subscribed FeedsProvider и TelephonyProvider.

Вам как программисту будет наиболее интересен исходный код, содержащийся в файле `android.jar` (если вы предпочитаете загрузить всю платформу и построить ее самостоятельно, обратитесь к документации, расположенной по адресу http://source.android.com/download). Исходники для этого файла JAR можно скачать, перейдя по следующей ссылке: http://git.source.android.com/?p=platform/frameworks/base.git;a=snapshot;h=HEAD;sf=tgz.

Эту универсальную ссылку можно использовать для загрузки проектов Git. В Windows этот файл можно распаковать при помощи `pkzip`. Хотя можно скачать

и распаковать исходники, удобнее будет просто просмотреть эти файлы онлайн, если вам не требуется производить отладку кода при помощи IDE. В Git можно выполнить и такую работу. Например, можно просмотреть исходные файлы архива android.jar, перейдя по ссылке http://android.git.kernel.org/?p=platform/frameworks/base.git;a=summary.

Правда, посетив эту страницу, вам еще придется поработать. Выберите из раскрывающегося списка параметр grep и введите в поисковую строку какой-нибудь текст. Чтобы открыть файл исходника у себя в браузере, щелкните на названии одного из найденных файлов. Данная функция позволяет быстро просмотреть исходный код.

Файл, который вы ищете, может отсутствовать в каталоге frameworks/base или в проекте. В таком случае найдите список проектов по адресу http://android.git.kernel.org/ и просмотрите их все по порядку.

Действие grep не распространяется на все проекты, поэтому вы должны знать, к какой функции Android относится конкретный проект. Например, графические библиотеки проекта Skia расположены по следующему адресу: http://android.git.kernel.org/?p=platform/external/skia.git;a=summary.

В файле SkMatrix.cpp содержится исходный код для матрицы преобразований, которая используется при анимации: http://android.git.kernel.org/?p=platform/external/skia.git;a=blob;f=src/core/SkMatrix.cpp.

Резюме

В этой главе мы хотели вызвать у вас интерес к Android. Вы узнали, что программирование в Android осуществляется на языке Java и что продвижением работ над Android занимается Open Handset Alliance. Мы рассмотрели, как мобильные устройства стали сравнимы в функциональном отношении с компьютерами общего назначения, и сделали обзор виртуальной машины Dalvik VM, которая позволяет задействовать сложный фреймворк на мобильном устройстве с ограниченными возможностями.

Мы также рассмотрели принцип работы Android и сравнили его с Java ME, исследовали программный стек Android и получили представление о концепциях, используемых при программировании в Android (эти концепции будут подробно рассмотрены в следующих главах). Вы познакомились с некоторыми примерами кода и узнали, где можно найти и скачать исходный код Android.

Мы надеемся, что в этой главе смогли убедить вас в том, что вы сможете успешно писать программы для платформы Android, не испытывая особых проблем. Приглашаем вас в путь, который продлится до последней страницы этой книги и в конце которого вы будете глубоко понимать инструментарий для разработки в Android.

2 Приступая к работе

В предыдущей главе мы сделали обзор истории Android и указали, какие концепции будут рассмотрены в остальных главах книги. Наверное, вы уже готовы приступить к работе с кодом. Мы покажем вам, с чего начать, чтобы вы могли писать код при помощи инструментария для разработки программ в Android (SDK) и поможем вам настроить среду разработки. Далее мы проведем вас через программу `Hello World!`, а затем препарируем приложение покрупнее. После этого мы расскажем о жизненном цикле приложения в Android и, наконец, коротко обсудим вопросы отладки приложений при помощи виртуальных устройств Android (Android Virtual Devices, AVD).

Чтобы создавать приложения для Android, вам понадобится инструментарий для разработки в Java SE (JDK), инструментарий для разработки в Android (Android SDK) и среда разработки. На самом деле приложения можно разрабатывать и в простейшем текстовом редакторе, но в этой книге мы будем пользоваться общедоступной программой Eclipse IDE. Android SDK требуют JDK версии 5 и выше (в примерах мы будем пользоваться JDK 6) и Eclipse 3.3 или выше (мы использовали Eclipse 3.5 или Galileo). В книге мы будем работать с Android SDK версии 2.0.

Наконец, чтобы работать было еще проще, пользуйтесь инструментами для разработки в Android (ADT). ADT — это подключаемый модуль (плагин) Eclipse, обеспечивающий создание приложений для Android при помощи Eclipse IDE. Все примеры этой книги были написаны при помощи Eclipse IDE и инструмента ADT.

Настройка среды разработки

Чтобы создавать приложения Android, необходимо настроить среду разработки. В данном разделе рассмотрим, как скачать JDK 6, Eclipse IDE, Android SDK и инструменты для разработки в Android (ADT). Кроме того, мы поможем вам сконфигурировать Eclipse для создания приложений Android.

Android SDK совместимы с Windows (XP, Vista и 7), Mac OS X (только Intel) и Linux (только Intel). В этой главе мы расскажем, как настроить среду разработки для всех этих платформ (что касается Linux, мы рассмотрим настройку среды разработки в дистрибутиве Ubuntu). В следующих главах мы не будем специально обсуждать межплатформенные различия.

Скачивание JDK 6

При разработке вам не обойтись прежде всего без инструментария для Java SE. Android SDK требуют JDK 5 или выше; в данной книге примеры создавались при помощи JDK 6. Если вы работаете в Windows, скачайте JDK 6 с сайта Sun (http://java.sun.com/javase/downloads/) и установите его. Вам понадобится только инструментарий для разработки в Java SE, а комплекты (bundles) — нет. Для Mac OS X, скачайте JDK с сайта Apple (http://developer.apple.com/java/download/), выберите файл, подходящий для вашей версии Mac OS, и установите его. Чтобы установить JDK для Linux, откройте окно терминала и введите в него следующую строку:

```
sudo apt-get install sun-java6-jdk
```

Так вы установите JDK и все зависимости (dependencies), например среду исполнения Java (JRE).

Далее установим переменную окружения JAVA_HOME, чтобы указать каталог для установки JDK. В системе Windows XP для этого сделайте следующее: нажмите кнопку **Пуск**, щелкните правой кнопкой на строке **Мой компьютер** и выберите команду **Свойства**. В открывшемся окне перейти на вкладку **Дополнительно**, а в ней выберите **Переменные окружения**. Нажмите **Создать**, чтобы добавить переменную, или **Правка**, чтобы изменить уже существующую переменную. JAVA_HOME будет иметь значение вида C:\Program Files\Java\jdk1.6.0_16. В Windows Vista и Windows 7 этапы перехода в окно переменных окружения немного отличаются. Нужно нажать **Пуск**, щелкнуть правой кнопкой мыши на строке **Компьютер**, выбрать команду **Свойства**, здесь нажать ссылку **Дополнительные настройки системы** и выбрать **Переменные окружения**. Далее для изменения переменной окружения JAVA_HOME выполните те же операции, что и в Windows XP. В Mac OS X переменная JAVA_HOME устанавливается в файле .profile в каталоге **HOME**. Отредактируйте или создайте файл .profile и добавьте в него следующую строку:

```
export JAVA_HOME=path_to_JDK_directory,
```

где path_to_JDK_directory может иметь значение /Library/Java/Home.

В Linux отредактируйте файл .profile и добавьте в него ту же строку, что и в примере с Mac OS X выше. Но значение пути будет иметь вид типа /usr/lib/jvm/java-6-sun.

Скачивание Eclipse 3.5

Когда JDK будет установлен, вы сможете скачать Eclipse IDE для разработчиков Java (вам не нужна версия для Java EE; она будет работать, но эта версия значительно объемнее, чем нам требуется, и содержит некоторые элементы, которые не будут рассматриваться в этой книге). В примерах этой книги используется Eclipse 3.5 (в среде Windows). Все версии Eclipse можно скачать с http://www.eclipse.org/downloads/. Дистрибутив Eclipse находится в файле ZIP, который можно распаковать практически в любой каталог. В Windows архив проще всего распаковать в C:\, в результате чего имеем каталог C:\eclipse, в котором находится файл eclipse.exe. В Mac OS X архив можно распаковать в **Applications**, а в Linux — в домашний каталог

(Home). Исполняемый файл Eclipse на всех платформах располагается в каталоге Eclipse.

При первом запуске Eclipse система потребует указать расположение рабочего пространства. Чтобы было проще, вы можете выбрать несложное расположение, например C:\android. Если компьютер используется совместно с другими пользователями, каталог рабочего пространства (workspace folder) должен находиться где-нибудь в подкаталоге Home.

Скачивание Android SDK

Для создания приложений для Android вам понадобится Android SDK. В SDK есть эмулятор, поэтому, чтобы разрабатывать приложения Android, вы можете обойтись мобильным устройством и без этой операционной системы. Мы разрабатывали образцы приложений для этой книги в Windows XP.

Android SDK можно скачать с http://developer.android.com/sdk. Android SDK поставляются в файле ZIP, подобно Eclipse, поэтому вам потребуется распаковать архив в подходящее место. В Windows распакуйте файл в удобное место (мы использовали диск C:), после чего у вас должен получиться каталог вида :\android-sdk-windows, в котором будут содержаться файлы, показанные на рис. 2.1. В Mac OS и Linux вы можете распаковать файл в домашний каталог (Home).

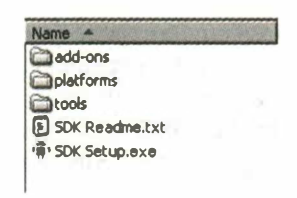


Рис. 2.1. Содержимое Android SDK

В Android SDK есть каталог инструментов, который должен быть указан в PATH. Давайте добавим его, а если вы занимаетесь обновлением системы, убедимся, что путь указан верно. Кроме того, добавим для JDK каталог bin — он пригодится нам в дальнейшем. В Windows нам нужно вернуться в окно переменных окружения, как это описано выше. Измените переменную PATH, добавив в конце точку с запятой (;), а после этого символа укажите путь в каталог инструментов Android SDK, за которым будет стоять еще одна точка с запятой, и далее — %JAVA_HOME%\bin. Закончив, нажмите OK. В Mac OS и Linux измените файл .profile и добавьте в переменную PATH путь к каталогу инструментов Android SDK. Здесь же следует указать путь к каталогу $JAVA_HOME/bin. Следующий вариант должен работать:

```
export PATH=$PATH:$HOME/android-sdk-linux_x86/tools:$JAVA_HOME/bin
```

В этой книге нам иногда потребуется прибегать к утилите командной строки. Данные программы будут входить в состав JDK или Android SDK. Указав каталоги в переменной PATH, мы избавим себя от необходимости указывать полные названия путей к файлам, чтобы выполнять их, а вместо этого будем запускать окно

инструментов (tools window). Мы подробнее рассмотрим это окно в следующих главах. В Windows для создания окна инструментов можно нажать кнопку Пуск, выбрать команду Выполнить, ввести в строку cmd и нажать OK. В Mac OS X выберите Terminal из каталога Applications в Finder или из Dock, если Terminal находится там. В Linux выберите Terminal из меню Applications ▸ Accessories.

И наконец, раз уж мы заговорили о межплатформенных различиях, возможно, вам потребуется узнать IP-адрес вашей рабочей станции. Чтобы сделать это в Windows, запустите окно инструментов и введите в него команду ipconfig. В результатах будет содержаться запись о IPv4 (или подобное), за которым будет следовать ваш IP-адрес. IP-адрес выглядит примерно так: 192.168.1.25. В Mac OS X и Linux откройте окно инструментов и используйте команду ifconfig. IP-адрес будет стоять вслед за обозначением (label) "inet addr". Вы увидите сетевое соединение под названием localhost или lo. IP-адрес такого соединения — 127.0.0.1. Это особое сетевое соединение, используемое операционной системой, но ваша рабочая станция будет иметь другой IP. Задайте его сами.

Установка инструментов разработки для Android (ADT)

Теперь нам необходимо установить ADT, плагин для Eclipse, который поможет вам при написании приложений Android. В частности, ADT интегрируется с Eclipse для обеспечения возможностей создания, тестирования и отладки приложений Android. Для установки следует использовать функцию Eclipse Install New Software (Установить новую программу). Если вы обновляете ADT, обратите внимание на команды, идущие за командами установки. Для начала запустите Eclipse IDE и выполните следующие шаги.

1. Откройте элемент меню Help (Помощь) и выберите команду Install New Software (Установить новую программу). В предыдущих версиях Eclipse эта функция называлась Software Updates (Обновления программ).
2. Выберите поле Work with (Работать с), введите в него https://dlssl.google.com/android/eclipse/ и нажмите клавишу Enter. Eclipse установит контакт с сайтом и заполнит список, который будет выглядеть так, как на рис. 2.2.
3. Вы должны видеть запись Developer Tools (Инструменты разработчика) с двумя подчиненными узлами: Android DDMS и Android Development Tools. Выберите родительский узел Developer Tools (Инструменты разработчика), убедитесь, что подчиненные узлы также выбраны, и нажмите кнопку Next (Далее). Версии, с которыми вам придется работать, наверняка будут новее, чем эта, и в SDK будут все обсуждаемые здесь функции.
4. Теперь необходимо будет подтвердить два инструмента, выбранные для установки. Снова нажмите Next (Далее).
5. Программа попросит вас просмотреть лицензию ADT, а также лицензии инструментов, необходимых для установки ADT. Прочитайте лицензионные соглашения, выберите I accept (Принимаю), а затем нажмите Finish (Готово).

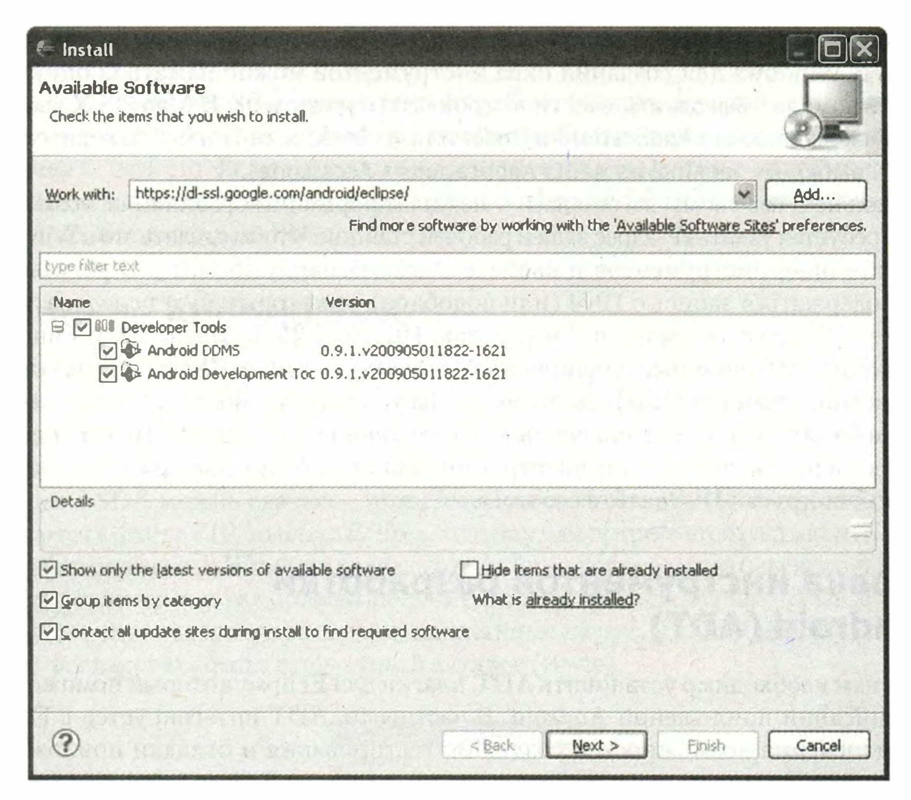


Рис. 2.2. Установка ADT при помощи функции Eclipse Install New Software (Установить новую программу)

Затем Eclipse скачает ADT и установит его. Будет нужно перезапустить Eclipse, чтобы плагин отобразился в IDE.

Если у вас в Eclipse уже установлены более ранние версии ADT, перейдите в меню Eclipse Help (Помощь) и выберите Check for updates (Проверить наличие обновлений). Вы должны увидеть новую версию ADT и сможете провести установку в соответствии с указаниями, данными выше, начиная с шага 3.

И наконец, чтобы активировать функционал ADT в Eclipse, нужно указать на Android SDK. Откройте меню Window (Окно) и выберите Preferences (Настройки) (В Mac OS X настройки находятся в меню Eclipse.) В диалоговом окне Preferences (Настройки) выберите узел Android и укажите в поле SDK Location (Расположение SDK) путь к Android SDK (рис. 2.3), а затем нажмите кнопку Apply (Применить). Обратите внимание — вы видите диалоговое окно, в котором предлагается отсылать статистику использования Android SDK в Google. Вы должны сами решить, согласны ли вы на это. Нажмите OK, чтобы закрыть окно Preferences (Настройки).

Когда вы впервые устанавливаете Android SDK, инструментарий не связан ни с одной из версий платформ. Если бы ситуация обстояла иначе, вы бы увидели версии платформ в окне настроек Android после настройки расположения SDK, как это показано на рис. 2.3. В Eclipse перейдите в Window ▸ Android SDK (Окно ▸ Android SDK) и AVD Manager (Диспетчер виртуальных устройств Android),

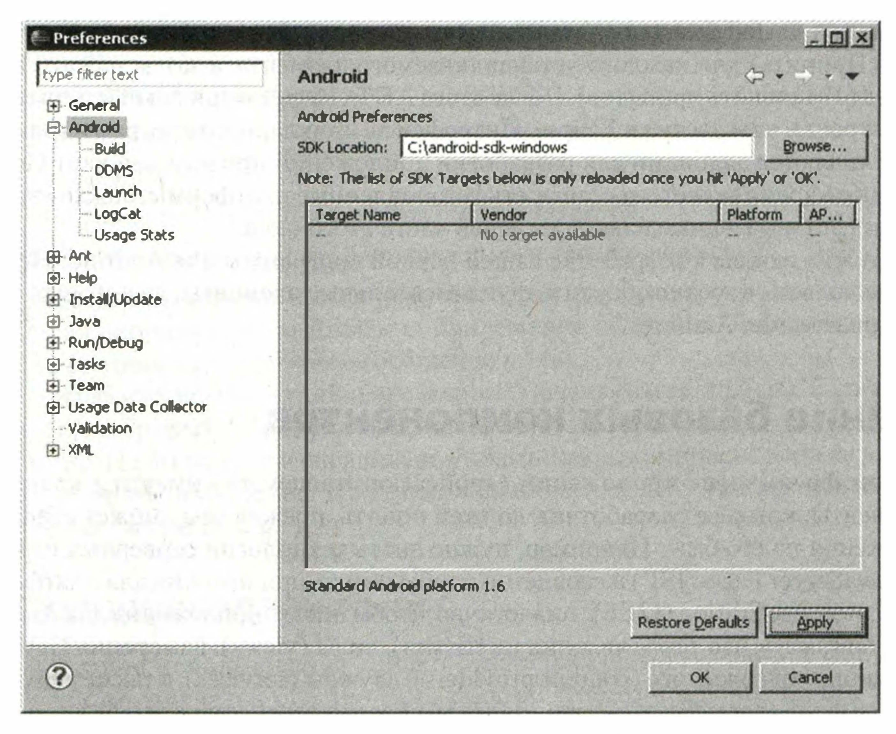


Рис. 2.3. Указание на Android SDK

выберите **Available Packages** (Доступные пакеты), после этого укажите источник **https://dl-ssl.google.com/android/repository/repository.xml**, затем задайте платформы и аддоны, которые вам нужны (например, Android 2.0) (рис. 2.4).

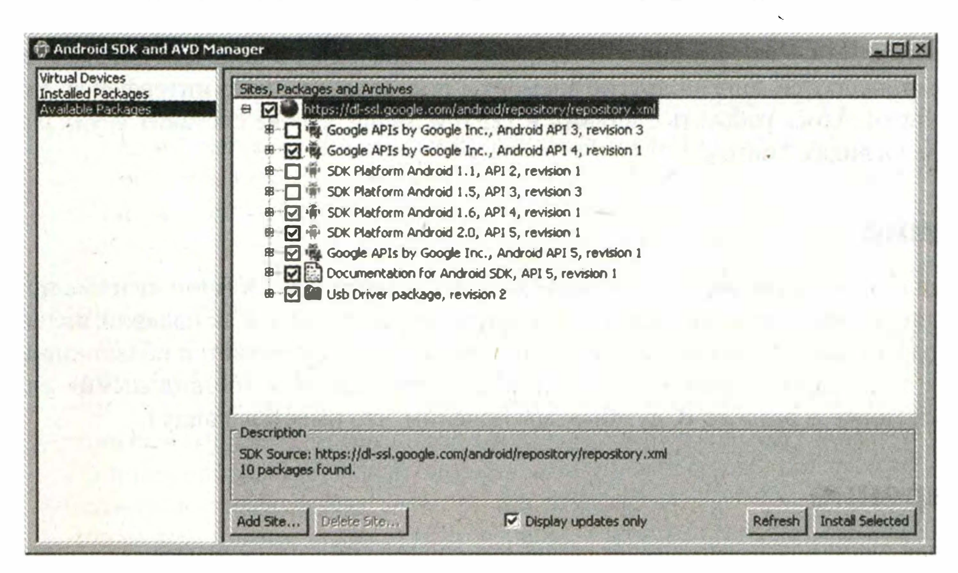


Рис. 2.4. Добавление платформ в Android SDK

Нажмите Install Selected (Установить выбранное). Нужно будет щелкнуть на Accept (Принять) для каждого устанавливаемого элемента, а затем нажать Install Accepted (Установить принятое). После этого ADT скачает ваши пакеты и платформы и откроет к ним доступ в Eclipse. Интерфейсы прикладного программирования Google являются аддонами для разработки приложений при помощи карт Google. Вы в любое время можете просмотреть установленные платформы, нажав Installed Packages (Установленные пакеты) в левой части этого окна.

Вы почти готовы к разработке вашей первой программы для Android. Но сначала мы должны коротко обсудить фундаментальные элементы, на которых основаны приложения Android.

Изучение базовых компонентов

В любом фреймворке приложений (application framework) имеются ключевые компоненты, которые разработчик должен понять, прежде чем сможет создавать приложения на его базе. Например, нужно знать технологии серверных страниц Java (JavaServer Pages, JSP) и сервлетов, чтобы писать приложения для платформы Java 2 Enterprise Edition (J2EE). Аналогично, чтобы писать приложения для Android, нужно понимать, что такое явления (activities), виды (views), намерения (intents), поставщики содержимого (content providers), службы (services), а также знать, как строится файл `AndroidManifest.xml`. Далее мы кратко рассмотрим эти базовые компоненты, а по ходу изложения материала обсудим каждый из них более подробно.

Вид

Виды — это базовые составляющие пользовательского интерфейса (UI), из которых строятся все его элементы. Виды иерархичны, и система может собирать одни виды из других. Вид — это и кнопка, и текстовое обозначение, и поле для ввода текста; видами являются многие другие элементы пользовательского интерфейса. Если вам приходилось работать с видами в J2EE и Swing, вам не составит труда разобраться в видах Android.

Явление

Явление — это компонент пользовательского интерфейса. Обычно явление представляет собой отдельное окно (экран) вашего приложения. Как правило, явление содержит один или несколько видов, но это условие не является обязательным. Более того, другие компоненты Android по сути являются «безвидовыми» явлениями (ниже, в разделе «Службы», мы поясним, что имеется в виду).

Намерение

Данная сущность представляет собой намерение (intention) выполнить определенное действие. В намерении объединено несколько компонентов; их проще

понять на примерах. Намерения можно использовать для выполнения таких задач, как:

- широковещательная передача сообщения;
- запуск службы;
- запуск явления;
- отображение веб-страницы или списка контактов;
- набор телефонного номера или ответ на телефонный вызов.

Намерения не всегда инициируются вашим приложением — они также используются системой для уведомления приложения об определенных событиях (например, о приходе текстового сообщения).

Намерения могут быть явными (explicit) и неявными (implicit). Если вы просто хотите отобразить URL, система сама решит, какой компонент выполнит это намерение. Но вы можете и специально указать, какой компонент должен выполнять определенное намерение. Намерения слабо связывают действие и обработчик действия (action handler).

Поставщик содержимого

Совместное использование данных несколькими приложениями мобильного устройства является обычной практикой. Поэтому в Android для этой цели предусмотрен стандартный механизм. Приложения могут совместно использовать данные (например, список контактов), не открывая доступ к памяти, внутренней структуре и способам реализации данных. При помощи поставщиков содержимого вы можете предоставлять свои данные другим приложениям и позволять своим программам использовать данные других приложений.

Службы

Службы в Android напоминают службы, которые используются в Windows и на других платформах — это фоновые процессы, которые потенциально могут работать в течение очень долгого времени. В Android различают два типа служб: локальные (local) и удаленные (remote). Локальные службы — это компоненты, доступ к которым открыт только для приложения, запустившего данную службу. И наоборот, удаленные службы могут использоваться любыми приложениями, работающими на данном устройстве.

Службой является, например, компонент почтового клиента, проверяющий, нет ли новых сообщений. Такая служба может быть локальной, если ее не используют другие приложения, работающие на данном устройстве. Если служба применяется несколькими приложениями, она должна быть реализована как удаленная. В главе 8 будет показана разница между двумя этими типами служб, которая заключается в противопоставлении `startService()` и `bindService()`.

Вы можете применять имеющиеся службы, а также создавать собственные, расширяя класс `Service`.

AndroidManifest.xml

Файл AndroidManifest.xml похож на файл web.xml из мира J2EE. В этом файле определяются содержание и способы работы приложения. Например, здесь перечисляются явления и службы вашего приложения, а также права, необходимые для его работы.

Виртуальные устройства Android

Виртуальное устройство Android (Android Virtual Device, AVD) позволяет разработчику тестировать свои приложения, не имея под рукой телефона с Android. Можно создавать AVD с различными видами конфигурации, чтобы эмулировать различные типы реальных телефонов.

Hello World!

Итак, мы готовы написать нашу первую программу для Android. Начнем с самого простого — с Hello World!. Чтобы создать каркас программы, выполните следующее.

1. Запустите Eclipse и выполните File ▸ New ▸ Project (Файл ▸ Новый ▸ Проект). В диалоговом окне New Project (Новый проект) выберите Android и нажмите Next (Далее). После этого вы перейдете в диалоговое окно New Android Project (Новый проект Android), которое показано на рис. 2.5. Команда Project Android (Проект Android) может содержаться в меню New (Создать) программы Eclipse, в таком случае можете выбрать ее прямо оттуда. Кроме того, можно воспользоваться кнопкой New Android Project (Новый проект Android), расположенной на панели инструментов.
2. В соответствии с рис. 2.5 укажите имя проекта HelloAndroid, имя приложения — HelloAndroidApp, имя пакета — com.androidbook и имя создаваемого явления — HelloActivity. Обратите внимание — для настоящего приложения нужно выбирать информативное имя, так как именно оно будет выводиться на панели с названиями приложений. Также обратите внимание на то, что задаваемое по умолчанию расположение проекта является производным от места расположения рабочего пространства Eclipse. В данном случае рабочим пространством Eclipse является C:\android, и мастер новых проектов добавляет имя нового приложения к месту расположения рабочего пространства, в итоге получается C:\android\HelloAndroid\. Наконец, версия Min SDK имеет значение 4, и это означает, что для работы вашего приложения требуется Android версии 1.6 или выше.
3. Нажмите кнопку Finish (Готово), в результате чего ADT создаст основу проекта. Теперь откройте файл HelloActivity.java в каталоге src и измените метод onCreate() следующим образом:

```
/** Вызывается при первом создании явления. */
   @Override
```

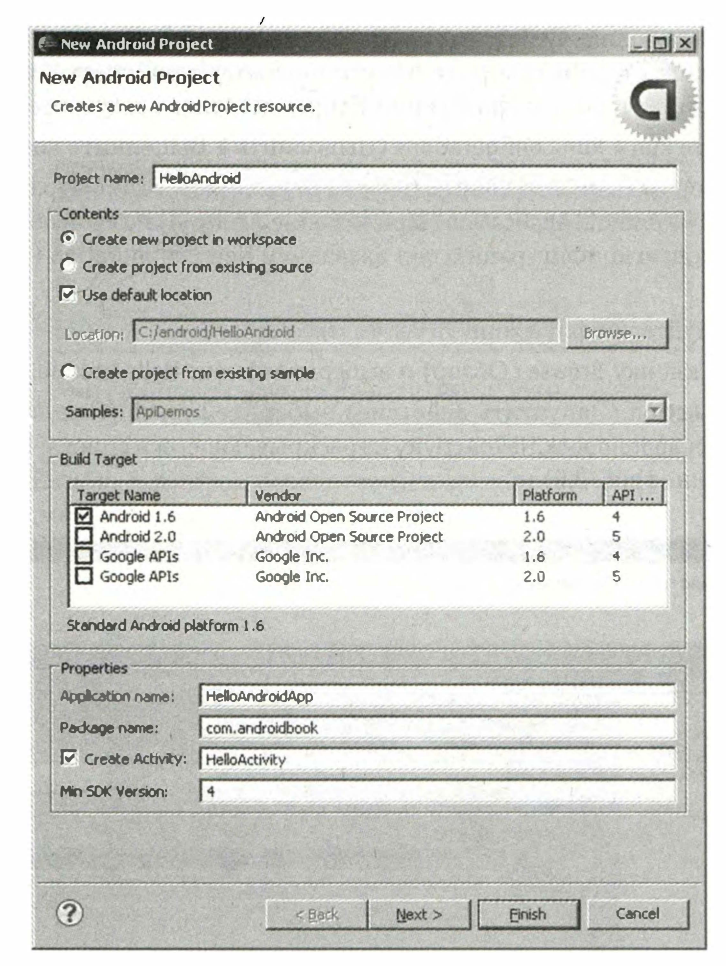


Рис. 2.5. Использование мастера новых проектов (New Project Wizard) при создании приложения для Android

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    /** создает TextView и записывает Hello World! */
    TextView tv = new TextView(this);
    tv.setText("Hello World!");
    /** устанавливает content view для TextView */
    setContentView(tv);
}
```

Eclipse должна автоматически добавить оператор (statement) `import` в `android.widget.TextView`. Вы можете нажать символ +, идущий после первого оператора `import`, чтобы увидеть все такие операторы. Если оператор `import` не добавляется автоматически, добавьте его сами. Сохраните файл `HelloActivity.java`.

Чтобы запустить приложение, следует создать стартовую конфигурацию Eclipse. Кроме того, вам понадобится виртуальное устройство, на котором можно будет

запустить приложение. Сейчас мы кратко рассмотрим эти этапы, а более подробно изучим их, когда будем говорить о виртуальных устройствах Android (AVD). Для создания стартовой конфигурации Eclipse сделайте следующее.

1. Выполните Run ▸ Run Configurations (Выполнить ▸ Выполнить конфигурацию).
2. В диалоговом окне Run Configurations (Выполнить конфигурацию) дважды щелкните на Android Application (Приложение Android) в левой области. Мастер добавит новую конфигурацию под названием New Configuration (Новая конфигурация).
3. Переименуйте конфигурацию в RunHelloWorld.
4. Нажмите кнопку Browse (Обзор) и выберите проект HelloAndroid.
5. В Launch Action (Запустить действие) выберите Launch (Запустить), а затем строку com.androidbook.HelloActivity из раскрывающегося списка. Появится диалоговое окно (рис. 2.6).

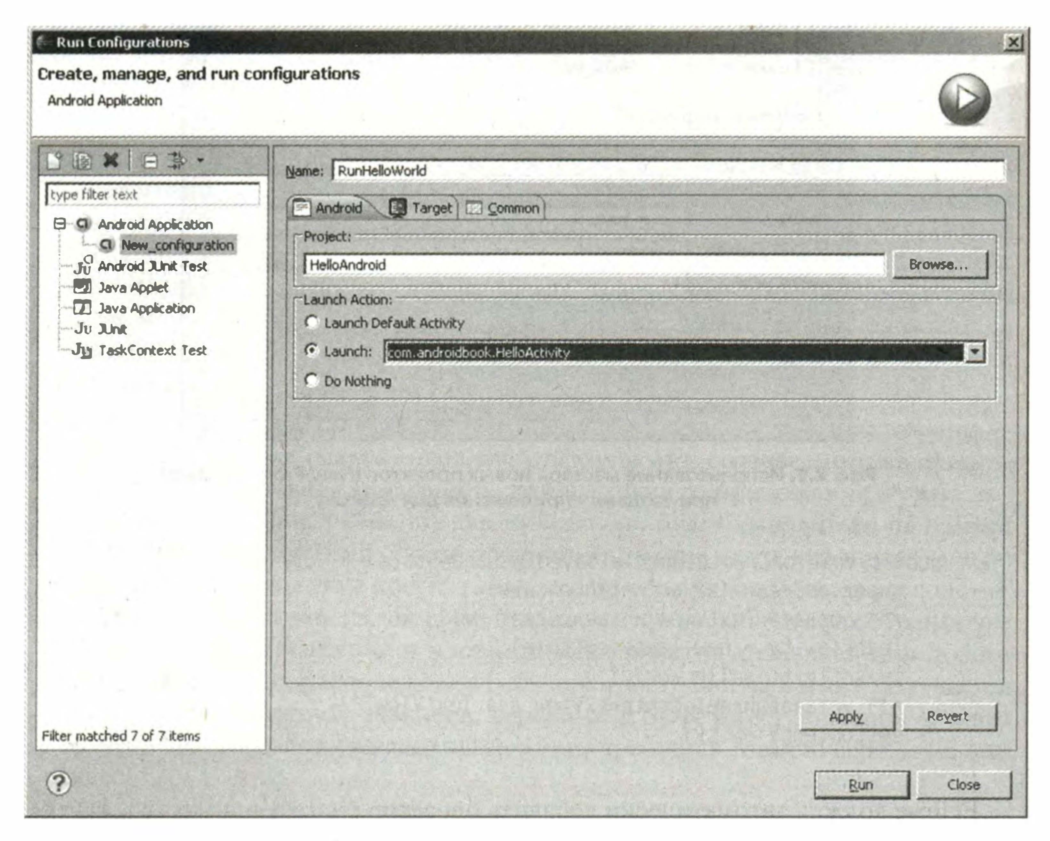


Рис. 2.6. Создание стартовой конфигурации Eclipse для запуска программы Hello World!

6. Нажмите Apply (Применить), а затем Run (Выполнить). Мы почти у цели. Eclipse готова запустить вашу программу, но ей требуется для этого соответствующее

устройство. На рис. 2.7 показано предупреждение о том, что в системе не найдено подходящих платформ (no compatible targets), после чего система предлагает вам создать устройство. Нажмите **Yes** (Да).

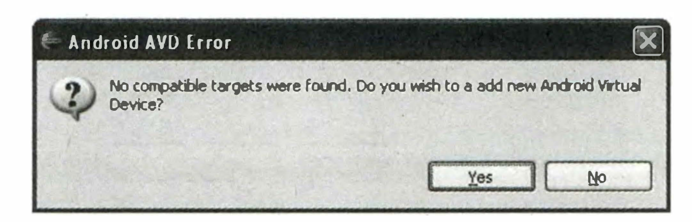


Рис. 2.7. Предупреждение Eclipse о совместимых платформах и запрос на создание нового AVD

7. Вы увидите окно, в котором перечислены имеющиеся AVD (рис. 2.8). Обратите внимание, что это же окно мы видели и ранее — на рис. 2.4. Нам нужно добавить виртуальное устройство для нового приложения. Нажмите кнопку **New** (Новое).

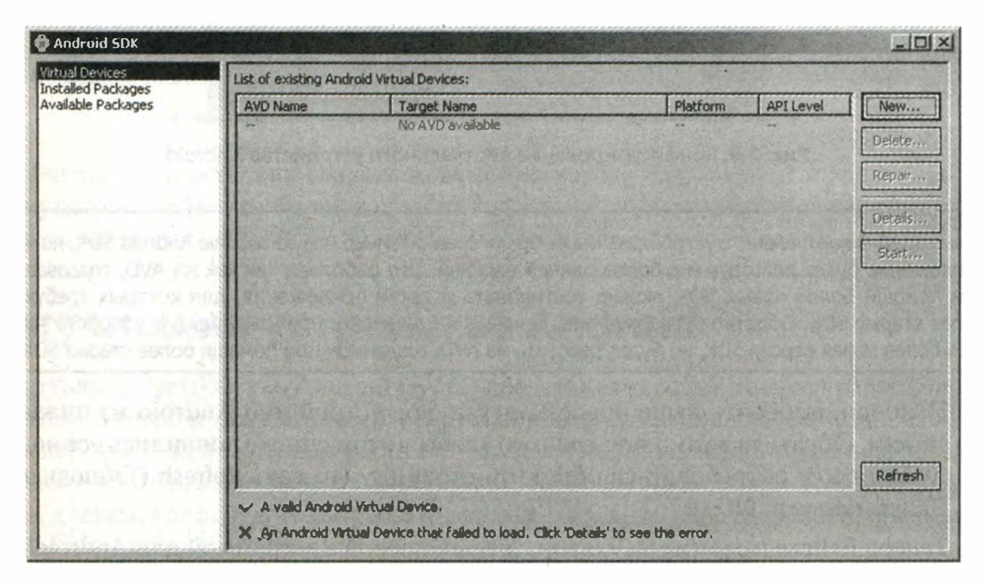


Рис. 2.8. Имеющиеся виртуальные устройства Android

8. Заполните форму **Create new AVD** (Создать новое AVD) так, как это показано на рис. 2.9. Установите для **Name** (Имя) — `DefaultAVD`, для **Target** (Цель) выберите **Android 2.0 — API Level 5**, задайте для карты памяти (**SD Card**) значение 32 (это соответствует 32 Мбайт), а для **Skin** (Оболочка) оставьте заданное по умолчанию разрешение HVGA (480 × 320 пикселов). Нажмите **Create AVD** (Создать AVD). Eclipse выдаст подтверждение о том, что виртуальное устройство Android создано успешно. Закройте окно Android SDK, нажав **OK**.

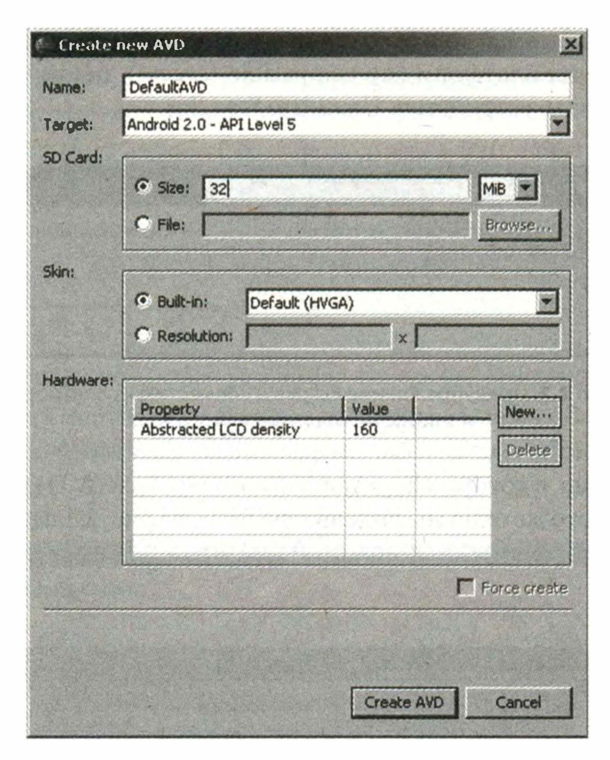


Рис. 2.9. Конфигурирование виртуального устройства Android

ПРИМЕЧАНИЕ

Для создания виртуального устройства мы выбрали сравнительно новую версию Android SDK, но наше приложение будет работать и с более ранней версией. Это работает, так как на AVD, создаваемых при помощи более новых SDK, можно тестировать и такие приложения, для которых требуются более старые SDK. Обратное утверждение, конечно же, неверно: приложение, для которого требуется более новая версия SDK, не будет работать на AVD, созданном при помощи более старых SDK.

9. Наконец, выберите ваше новое виртуальное устройство Android из нижнего списка. Обратите внимание: для того чтобы в этом списке появились все новые AVD, может потребоваться обновить страницу (нажать Refresh (Обновить)). Далее нажмите OK.
10. Теперь Eclipse запустит эмулятор с вашей первой программой для Android!

ПРИМЕЧАНИЕ

На то, чтобы смоделировать процесс загрузки устройства, эмулятору может потребоваться около минуты. После запуска вы увидите, что в эмуляторе выполняется программа HelloAndroidApp, как это показано на рис. 2.10. Кроме того, не забывайте, что при старте этой программы эмулятор запускает в фоновом режиме и другие приложения, поэтому время от времени система может выводить предупреждение или сообщение об ошибке. Если появляется сообщение об ошибке, его можно проигнорировать, чтобы эмулятор перешел к следующему этапу запуска. Например, если вы запускаете эмулятор и видите сообщение типа «приложение abc не отвечает», можно либо дождаться запуска этого приложения, либо приказать эмулятору принудительно закрыть эту программу. Как правило, следует подождать, чтобы эмулятор запускал программу аккуратно.

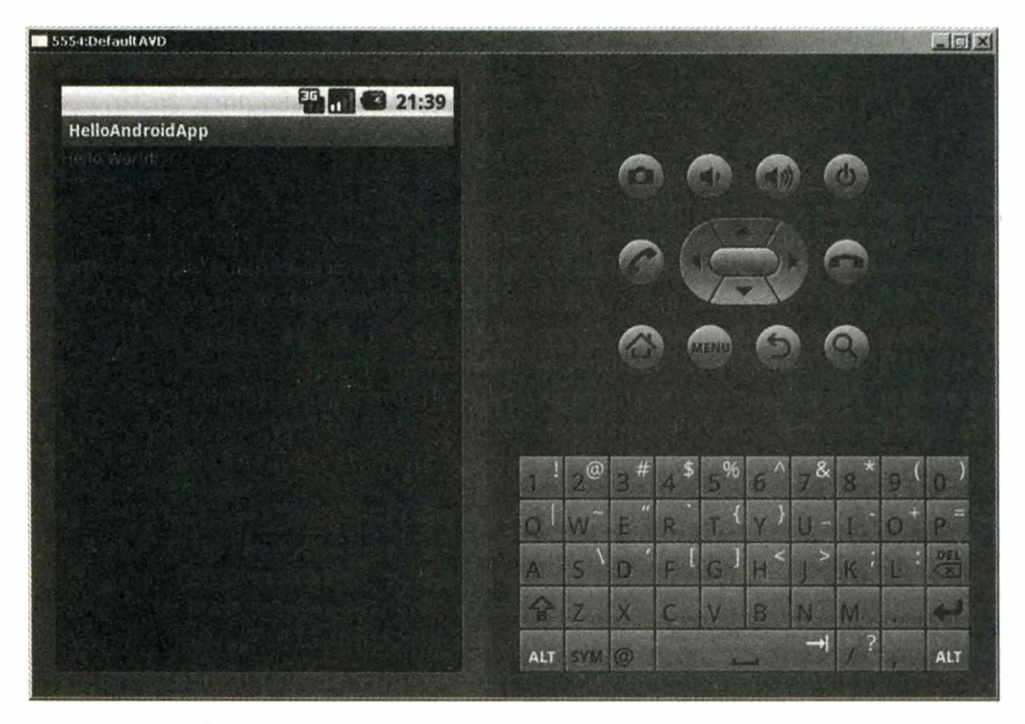


Рис. 2.10. Программа HelloAndroidApp работает в эмуляторе

Теперь вы знаете, как создать новую программу для Android и запустить ее в эмуляторе. Далее мы более подробно изучим виртуальные устройства Android, а затем углубимся в работу с артефактами и структурой приложений Android.

Виртуальные устройства Android

Виртуальное устройство Android (AVD) представляет собой конфигурацию определенного устройства. Например, у вас может быть AVD, являющееся более ранней версией Android, на котором работает версия SDK 1.5 с картой памяти 32 Мбайт. Смысл в том, что вы можете создавать такие AVD, которые собираетесь поддерживать, а затем, когда будете разрабатывать и тестировать свое приложение, указать эмулятору на одно из таких виртуальных устройств. Вы можете с легкостью задать (а затем при необходимости изменить) устройство, которое должен использовать эмулятор, и благодаря этому тестирование различных вариантов конфигурации не составляет никакого труда. Выше было показано, как создать AVD при помощи Eclipse. Чтобы создать в Eclipse больше устройств, перейдите в Window ▸ Android SDK (Окно ▸ Android SDK) и далее в диспетчер виртуальных устройств Android (AVD Manager), а затем щелкните в левой части окна Virtual Devices (Виртуальные устройства). Кроме того, вы можете создать AVD при помощи командной строки. Ниже рассказано, как это сделать.

Для создания AVD используется командный файл (batch file) под названием `android`, находящийся в каталоге `tools` (`c:\android-sdk-windows\tools\`). Android по-

зволяет создать новое AVD, а также управлять имеющимися AVD. Например, вы можете просмотреть имеющиеся AVD, перемещать AVD и т. д. Чтобы просмотреть параметры, доступные для использования с Android, выполните `android -help`. Сначала давайте просто создадим виртуальное устройство Android.

По умолчанию AVD сохраняются в домашнем каталоге (на всех платформах), в папке, называемой `.android\AVD`. Если вы создали AVD для программы `Hello World!`, описанной выше, то найдете его именно здесь. Если вы хотите сохранять AVD (и управлять ими) в каком-либо другом месте, это также можно сделать. В данном примере давайте создадим каталог, в котором будет храниться образ AVD, например `c:\avd\`. Следующий этап — запуск файла `android` с последующим созданием AVD. Откройте окно инструментов и введите в него следующую команду (используйте правильный путь, чтобы сохранять файлы AVD для своей рабочей станции, а также правильное значение аргумента `t`, в зависимости от того, какую более раннюю версию платформы SDK вы установили):

```
android create avd -n OlderAVD -t 2 -c 32M -p C:\AVD\OlderAVD\
```

Параметры, переданные командному файлу, приведены в табл. 2.1.

Таблица 2.1. Параметры, передаваемые инструменту android.bat

Аргумент/команда	**Описание**
create avd	Приказывает инструменту создать AVD
N	Имя AVD
T	Нужная среда исполнения (target runtime). 1 соответствует Android 1.1, 2 — Android 1.5, 3 — Android 1.6 и т. д.
c	Размер карты памяти в байтах. K соответствует килобайтам, M — мегабайтам
p	Путь к сгенерированному AVD. Указывать не обязательно

После выполнения предыдущей команды создается AVD; вывод должен выглядеть примерно так, как на рис. 2.11. Обратите внимание: при запуске команды `create avd`, система задаст вопрос, хотите ли вы создать специальный аппаратный профиль. Пока ответьте на этот запрос no (нет), так как при ответе yes (да) система предложит вам множество парамеров для AVD, в частности размер экрана, наличие камеры и т. д.

```
C:\WINDOWS\system32\cmd.exe

C:\avd>android create avd -n OlderAVD -t 1 -c 32M -p C:\AVD\OlderAVD\
Android 1.1 is a basic Android platform.
Do you wish to create a custom hardware profile [no]no
Created AVD 'OlderAVD' based on Android 1.1

C:\avd>dir
 Volume in drive C has no label.
 Volume Serial Number is 2C84-1F11

 Directory of C:\avd

09/08/2009  07:33 PM    <DIR>          .
09/08/2009  07:33 PM    <DIR>          ..
09/08/2009  07:33 PM    <DIR>          OlderAVD
               0 File(s)              0 bytes
               3 Dir(s)  59,355,009,024 bytes free

C:\avd>
```

Рис. 2.11. Вывод android.bat при создании AVD

Даже если вы укажете для сохранения OlderAVD нестандартное расположение при помощи программы android.bat, в вашем домашнем каталоге, в папке .android/AVD, будет находиться файл OlderAVD.ini. И это хорошо, так как если вы вернетесь в Eclipse, выберете **Window ▸ Android SDK** (Окно ▸ Android SDK) и диспетчер AVD (**AVD Manager**), то увидите все свои AVD и сможете получить доступ к любому из них при запуске программ Android через Eclipse.

Еще раз обратимся к рис. 2.5. Для работы с программой Hello World! мы выбрали Android 1.6 и установили для версии Min SDK значение 4. Если выбрать Android 1.5 (предполагается, что вы его установили), для версии Min SDK устанавливается значение 3. При применении Android 2.0 для версии Min SDK устанавливается значение 5.

Не забывайте и о том, что при выборе API Google из списка версий SDK вы включаете в свое приложение функцию картографирования (mapping functionality), а при выборе версии Android 1.5 и выше этого не происходит. В предыдущих версиях инструментария SDK, вышедших до версии Android 1.5, классы, связанные с картографированием, входили в файл android.jar, а в Android 1.5 и выше эти файлы вынесены в отдельный архив, называемый maps.jar. При выборе интерфейса прикладного программирования для Google Min SDK по умолчанию будет иметь версию 5 (для Android 2.0) или 4 (для Android 1.6) и т. д, а плагин ADT будет включать в ваш проект файл maps.jar. Если вы пишете приложение, в котором будут использоваться классы, связанные с картографированием, выберите SDK Target для Google API. Обратите внимание на то, что необходимо добавить в файл AndroidManifest.xml запись об использовании картографической библиотеки (<uses-library android:name="com.google.android.maps" />). Подробнее мы рассмотрим этот вопрос в главе 7.

Изучение структуры приложения в Android

Размер и степень сложности приложений Android могут сильно различаться, но строение этих приложений будет схожим. На рис. 2.12 показана структура программы Hello World!, которую мы недавно написали.

В программах Android содержатся артефакты. Некоторые из них являются необходимыми, а другие — опциональными. В табл. 2.2 в обобщенной форме рассказано об элементах приложения Android.

Из табл. 2.2 видно, что приложение Android строится из трех основных компонентов: дескриптора приложения, набора различных ресурсов и исходного кода программы. Если абстрагироваться от файла AndroidManifest.xml, можно увидеть, что архитектура приложения Android очень проста: у нас есть немного бизнес-логики, реализованной в коде, а все остальное — это ресурсы. Такая базовая структура напоминает строение приложений в J2EE, где с ресурсами можно сравнить серверные страницы Java (JSP), бизнес-логика реализуется в виде сервлетов, а файл AndroidManifest.xml соответствует файлу web.xml.

Кроме того, можно сравнить модели разработки, применяемые в J2EE и в Android. Философия построения видов в J2EE связана с использованием языка разметки. В Android также применяется такой подход, но для разметки здесь используется XML. Такой подход удобен потому, что от вас не требуется жестко кодировать виды, применяемые в приложениях; внешний вид приложения и впечатление от него можно изменить, просто отредактировав разметку.

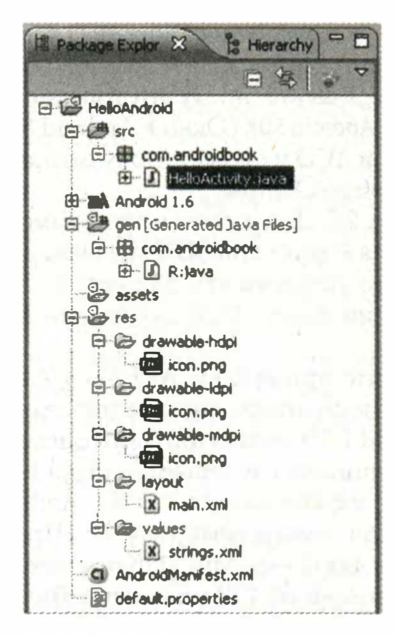


Рис. 2.12. Структура программы Hello World!

Таблица 2.2. Артефакты приложения в Android

Артефакт	Описание	Необходимость
AndroidManifest.xml	Файл описания приложения Android. В этом файле определяются явления, поставщики содержимого, службы и приемники намерений (intent receiver) данного приложения. Этот файл также можно использовать для декларативного объяснения прав доступа, требуемых для работы с приложением, и для предоставления специфических прав доступа другим приложениям, которые будут пользоваться службами данного. Кроме того, в этом файле может содержаться информация об инструментировании (instrumentation detail), которая может использоваться при тестировании данного или иного приложения	Да
src	Каталог, в котором содержится весь исходный код приложения	Да
assets	Произвольное собрание каталогов и файлов	Нет
res	Каталог, в котором содержатся ресурсы приложения. Данный каталог является родительским для drawable, anim, layout, menu, values, xml и raw	Да
drawable	Каталог, который содержит изображения или файлы дескрипторов изображений, используемые программой	Нет
anim	Каталог, содержащий файлы XML-дескрипторов, в которых описывается анимация, используемая программой	Нет
layout	Каталог, в котором содержатся виды данной программы. Виды программы создаются при помощи XML-дескрипторов, а не путем написания кода	Нет

Артефакт	Описание	Необходимость
menu	Каталог, содержащий файлы XML-дескрипторов, в которых описываются меню, используемые в программе	Нет
values	Каталог, содержащий другие ресурсы, используемые программой. Все ресурсы из этого каталога также определяются при помощи XML-дескрипторов. К числу ресурсов, которые находятся в этом каталоге, относятся строки, стили и цвета	Нет
xml	Каталог, содержащий дополнительные XML-файлы, используемые приложением	Нет
raw	Каталог, содержащий дополнительные данные — возможно, не в формате XML, — которые нужны для работы программы	

Необходимо также сказать о некоторых ограничениях, касающихся ресурсов. Во-первых, в Android поддерживается только линейный список файлов, находящийся в специальном каталоге, расположенном под `res`. Например, в Android не поддерживаются вложенные каталоги, находящиеся под `layout` (или иные каталоги, находящиеся под `res`). Во-вторых, есть определенное сходство между каталогом `assets` и каталогом `raw`, который находится под `res`. В обоих каталогах могут содержаться RAW-файлы, но те файлы, которые находятся в каталоге `raw`, считаются ресурсами, а находящиеся в `assets` — нет. Таким образом, файлы каталога `raw` локализуются, к ним открывается доступ через ID ресурса и т. д. Обратите внимание — поскольку файлы каталога `assets` не считаются ресурсами, в этом каталоге папки и файлы могут иметь произвольную иерархию (более подробно мы рассмотрим ресурсы в главе 3).

ПРИМЕЧАНИЕ

Вы, вероятно, уже поняли, насколько активно XML используется в Android. Как известно, XML — это достаточно громоздкий формат данных, поэтому возникает вопрос — зачем полагаться на XML, если мы пишем программу для устройства с ограниченными ресурсами? Но оказывается, что XML, создаваемый нами при разработке, скомпилирован в двоичный формат, и это делается при помощи инструмента Android Asset Packaging Tool (AAPT). Затем, когда ваше приложение будет установлено в устройство, файлы в устройстве также будут сохраняться в двоичном формате. Как только определенный файл потребуется программе для исполнения, он будет считан в двоичной форме и не будет преобразовываться обратно в XML. Таким образом, мы одновременно и пользуемся преимуществами XML, и обходимся сравнительно ограниченными ресурсами устройства.

Анализ приложения NotePad

Теперь вы знаете, как создать новое приложение в Android и запустить его в эмуляторе, но нам еще нужно познакомиться с артефактами, которые входят в состав приложения Android. В этом разделе мы рассмотрим программу NotePad (Блокнот), которая входит в состав инструментария для разработки в Android. По сложности NotePad занимает промежуточное положение между программкой `Hello World!` и полноценным приложением Android. По этой причине, проанализировав компо-

ненты NotePad, вы сможете составить достаточно точное впечатление о разработке программ для Android.

Загрузка и запуск приложения NotePad

В этом разделе мы рассмотрим, как загрузить программу NotePad в Eclipse IDE и запустить ее в эмуляторе. Перед тем как начать, оговоримся, что приложение NotePad можно использовать несколькими способами. Например, пользователь может создать новую запись, отредактировать или удалить имеющуюся запись, просмотреть список созданных записей. Когда пользователь запускает приложение, сохраненных записей еще нет, то есть пользователь видит вместо списка незаполненное пространство. Если нажать клавишу Menu (Меню), приложение выведет список действий, одно из которых — создание новой записи. После того как запись будет добавлена, пользователь может отредактировать или удалить ее, выбрав в меню соответствующую команду.

Выполните следующие шаги, чтобы загрузить образцы файлов NotePad в Eclipse IDE.

1. Запустите Eclipse.
2. Выполните команду File ▸ New ▸ Project (Файл ▸ Новый ▸ Проект).
3. В диалоговом окне New Project (Новый проект) выберите Android ▸ Android Project (Android ▸ Проект Android).
4. В окне Android Project (Проект Android) в поле Project Name (Название проекта) введите `NotesList`, выберите Create project from existing sample (Создать проект из имеющегося образца), далее нажмите Build Target of Android 2.0 (Выбрать в качестве целевой платформы Android 2.0) и в меню Samples (Образцы) найдите приложение NotePad. Обратите внимание — приложение NotePad располагается в каталоге `platforms\android-2.0\samples` инструментария для разработки в Android, который вы скачали ранее. После того как вы выберете NotePad, система считает файл `AndroidManifest.xml` и автоматически предварительно заполнит оставшиеся поля в диалоговом окне New Android Project (Новый проект Android) (рис. 2.13).
5. Нажмите кнопку Finish (Готово).

После этого приложение NoteList должно загрузиться в Eclipse IDE. Если Eclipse сообщает, что с этим проектом возникли определенные проблемы, попробуйте воспользоваться функцией Clean (Очистить) из меню Project (Проект) в Eclipse. Для запуска приложения нужно создать стартовую конфигурацию (мы уже делали это с программой `Hello World!`) или просто щелкнуть правой кнопкой мыши на названии проекта, выбрать Run as (Выполнить как) и указать Android Application (Приложение для Android). Таким образом, мы запустим эмулятор и установим в нем приложение. После того как эмулятор завершит загрузку (вы увидите в центре окна эмулятора дату и время), нажмите кнопку Menu (Меню), чтобы просмотреть программу NotePad. Поэкспериментируйте с разными функциями приложения, познакомьтесь с ним поближе.

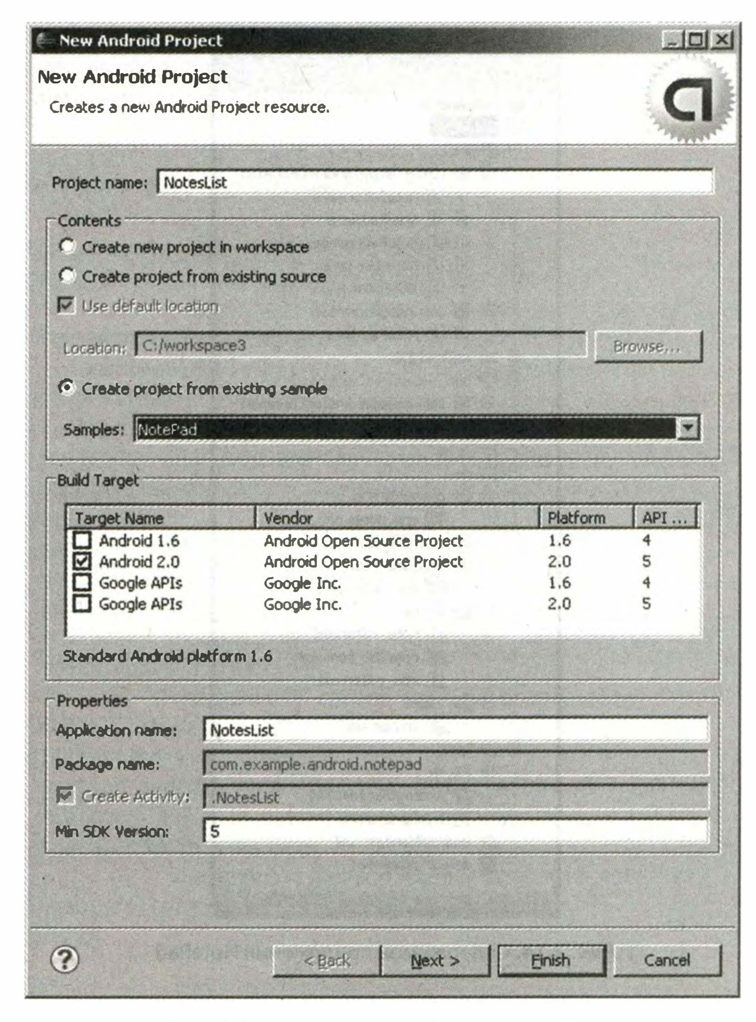


Рис. 2.13. Создание приложения NotePad

Подробный анализ приложения

Теперь изучим содержимое приложения NotePad (рис. 2.14).

Как видно на рис. 2.14, в программе содержится несколько файлов JAVA, несколько изображений PNG, три вида (в каталоге `layout`) и файл `AndroidManifest.xml`. Если бы это было приложение командной строки, нам нужно было бы найти класс, связанный с методом `Main`. Какой же компонент Android аналогичен методу `Main`?

В Android можно задать явление «точка входа в программу» (entry-point activity), которая также называется явлением верхнего уровня (top-level activity). Просмотрев файл `AndroidManifest.xml`, вы найдете в нем один поставщик и три вида явлений. Явление `NotesList` задает фильтр намерений для действия `android.intent.action.MAIN`

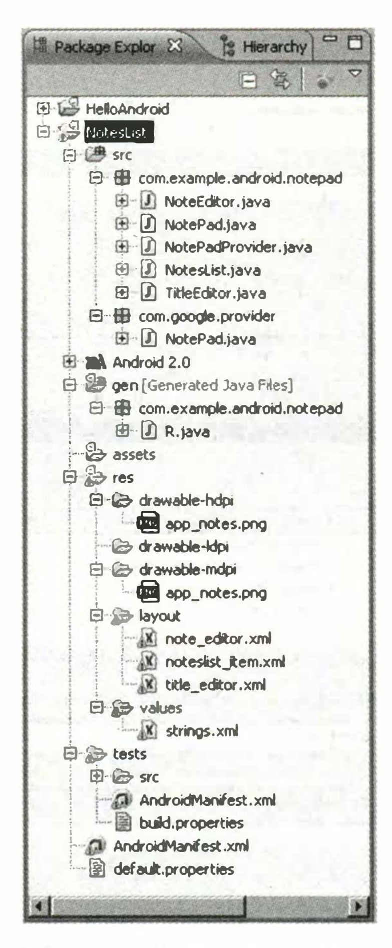


Рис. 2.14. Содержимое приложения NotePad

и для категории `android.intent.category.LAUNCHER`. Когда поступает запрос на запуск приложения Android, хост-машина загружает приложение и считывает файл `AndroidManifest.xml`. После этого компьютер ищет и запускает явление (или явления) с фильтром намерений, в котором указано действие `MAIN` с категорией `LAUNCHER`, как это показано ниже:

```
<intent-filter>
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

После того как хост-машина найдет явление, которое необходимо запустить, она должна соотнести заданное явление с определенным классом. Это делается путем комбинирования имени корневого пакета проекта (`root package name`) и имени явления. В нашем случае имеем `com.example.android.notepad.NotesList` (листинг 2.1).

Листинг 2.1. Файл AndroidManfiest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.notepad"
>
   <application android:icon="@drawable/app_notes"
      android:label="@string/app_name"
   >
      <provider android:name="NotePadProvider"
         android:authorities="com.google.provider.NotePad"
      />
      <activity android:name="NotesList"
               android:label="@string/title_notes_list">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <action android:name="android.intent.action.EDIT" />
            <action android:name="android.intent.action.PICK" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType=
                     "vnd.android.cursor.dir/vnd.google.note" />
         </intent-filter>
         <intent-filter>
            <action android:name="android.intent.action.GET_CONTENT" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType=
                     "vnd.android.cursor.item/vnd.google.note" />
         </intent-filter>
      </activity>
…
</manfiest>
```

Название корневого пакета приложения определяется как атрибут элемента <manifest> в файле AndroidManifest.xml, и каждое явление имеет атрибут name.

После того как будет определено явление для точки входа в программу, хост начинает выполнять явление, вызывая метод onCreate(). Рассмотрим NotesList.onCreate(), показанный в листинге 2.2.

Листинг 2.2. Метод onCreate

```
public class NotesList extends ListActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);

         setDefaultKeyMode(DEFAULT_KEYS_SHORTCUT);
         Intent intent = getIntent();
         if (intent.getData() == null) {
```

```
            intent.setData(Notes.CONTENT_URI);
        }

        getListView().setOnCreateContextMenuListener(this);

        Cursor cursor = managedQuery(getIntent().getData(),
                    PROJECTION, null, null,
                    Notes.DEFAULT_SORT_ORDER);
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
    R.layout.noteslist_item, cursor, new String[] { Notes.TITLE },
    new int[] { android.R.id.text1 });
    setListAdapter(adapter);
    }
}
```

Обычно в Android для запуска явлений используются намерения, но одно явление также может запускать другое. Метод onCreate() проверяет, есть ли данные (записи) у намерения конкретного явления. Если нет, метод задает URI (унифицированный идентификатор ресурса), чтобы найти информацию о намерении. В главе 3 вы увидите, что Android получает доступ к данным через поставщики содержимого, которые, в свою очередь, оперируют URI. В таком случае URI располагает достаточной информацией, чтобы получать данные из базы данных. Константа Notes.CONTENT_URI определена в Notepad.java как static final:

```
public static final Uri CONTENT_URI =
      Uri.parse("content://" + AUTHORITY + "/notes");
```

Класс Notes является внутренним (inner class) по отношению к классу Notepad. Пока отметим, что предыдущий URI приказывает поставщику содержимого получить все записи. Если бы URI выглядел следующим образом:

```
public static final Uri CONTENT_URI =
      Uri.parse("content://" + AUTHORITY + "/notes/11");
```

то потребляющий поставщик содержимого возвратил бы запись с ID, равным 11. Более подробно мы рассмотрим поставщики содержимого и унифицированные идентификаторы ресурсов в главе 3.

Класс NoteList дополняет класс ListActivity, в котором содержится информация о том, как отображать списковые данные. Элементы списка управляются внешним ListView (компонентом пользовательского интерфейса), который отображает записи в форме списка. После того как намерению явления будет присвоен URI, явление фиксируется как предназначенное для создания контекстных меню для записей. Если вы уже поэкспериментировали с приложением, то, наверное, заметили, что набор чувствительных к регистру элементов, отображаемых в меню, зависит от вашего выбора. Например, если вы выберете существующую запись, приложение предложит **Edit note** (Отредактировать запись) и **Edit title** (Отредактировать заголовок). Соответственно, если вы не выделите ни одной записи, приложение отобразит функцию **Add note** (Добавить запись).

Далее вы видите, как явление выполняет управляемый запрос и получает курсор для вывода результатов. При управляемом запросе Android может управлять по-

лученным курсором. В частности, это управление выражается в том, что если будет необходимо выгрузить из памяти или перезагрузить программу, то ни само приложение, ни действующее явление не будут заниматься при этом размещением курсора, а также загрузкой курсора и выгрузкой его из памяти. Параметры managedQuery(), показанные в табл. 2.3, достаточно интересны.

Таблица 2.3. Параметры Activity.managedQuery()

Параметр	Тип данных	Описание
URI	Унифицированный идентификатор ресурса	Унифицированный идентификатор ресурса поставщика содержимого
projection	Строка[]	Возвращаемый столбец (названия столбцов)
selection	Строка	Необязательный оператор where
selectionArgs	Строка[]	Аргументы для выборки, если запрос содержит символы ?
sortOrder	Строка	Порядок сортировки, который будет использоваться в результирующем наборе

Мы рассмотрим запрос managedQuery() и находящийся с ним на одном уровне иерархии запрос query() далее в этом разделе, а также в главе 3. Пока остановимся на том, что запрос в Android возвращает данные в форме таблицы. Параметр projection позволяет задать столбцы, которые будут нужны вам в такой таблице. Кроме того, вы можете сократить общий набор результатов и отсортировать набор результатов при помощи оператора сортировки SQL (например, asc или desc). Обратите также внимание на то, что запрос Android должен возвратить столбец с именем ID, предназначенный для поддержки поиска конкретных записей. Вы должны знать тип данных, возвращаемых поставщиком содержимого, — столбец может содержать string, int, binary и другие типы данных.

После того как запрос будет выполнен, возвращенный курсор передается конструктору SimpleCursorAdapter, который адаптирует записи, содержащиеся в наборе данных, к элементам пользовательского интерфейса (ListView). Рассмотрим подробнее параметры, переданные конструктору SimpleCursorAdapter:

```
    SimpleCursorAdapter adapter =
  new SimpleCursorAdapter(this, R.layout.noteslist_item,
cursor, new String[] { Notes.TITLE }, new int[] { android.R.id.text1 });
```

Особенно важен второй параметр — идентификатор вида, представляющего элементы в ListView. В главе 3 будет рассказано, что в Android предоставляется автоматически генерируемый вспомогательный класс, создающий ссылки на ресурсы вашего проекта. Этот вспомогательный класс называется R-класс, так как его имя — R.java. Когда вы компилируете проект, AAPT генерирует R-класс, основываясь на том, какие ресурсы содержатся в каталоге res. Например, вы можете поместить все строковые ресурсы в каталог values, и AAPT сгенерирует идентификатор public static для каждой строки. В общем виде в Android такая функция поддерживается для любых ресурсов. Например, при применении конструктора SimpleCursorAdapter явление NotesList передается в идентификаторе вида, который отображает элемент списка записей. Польза такого вспомогательного класса

заключается в том, что вам не приходится жестко кодировать ваши ресурсы, а проверка ссылок происходит уже во время компилирования. Иными словами, если определенный ресурс окажется удален, класс R потеряет с ним связь и любой код, ссылающийся на ресурс, компилироваться не будет.

Теперь рассмотрим еще одну важную концепцию Android, о которой мы упоминали выше: метод `onListItemClick()`, относящийся к `NotesList` (листинг 2.3).

Листинг 2.3. Метод onListItemClick

```
@Override
   protected void onListItemClick(ListView l, View v, int position,
                                             long id) {
      Uri uri = ContentUris.withAppendedId(getIntent().getData(), id);

      String action = getIntent().getAction();
      if (Intent.ACTION_PICK.equals(action) ||
Intent.ACTION_GET_CONTENT.equals(action)) {
         setResult(RESULT_OK, new Intent().setData(uri));
      } else {
         startActivity(new Intent(Intent.ACTION_EDIT, uri));
      }
   }
```

Вызов метода `onListItemClick()` происходит, когда пользователь выбирает запись в интерфейсе. На примере этого метода видно, как одно явление может запускать другое. Когда запись выбрана, метод создает для него идентификатор URI — для этого берется базовый URI, к которому прикрепляется ID выбранной записи. Затем этот URI передается `startActivity()` вместе с новым намерением. `startActivity()` — это один из вариантов запуска явления; этот метод запускает явление, но не сообщает о результатах после того, как явление завершится. Другой метод, позволяющий запустить явление, связан с использованием `startActivityForResult()`. При помощи такого метода можно запустить другое явление и зарегистрировать обратный вызов, который должен быть произведен после завершения выполнения явления. Например, вы можете воспользоваться `startActivityForResult()` для запуска такого явления, как выбор контакта, поскольку вы хотите установить контакт сразу же после того, как это явление будет выполнено.

Здесь нас может заинтересовать проблема взаимодействия пользователя с системой при работе с явлениями. Например, если выполняемое явление запускает другое явление и *то* явление также запускает новое явление и т. д, то с каким именно явлением будет взаимодействовать пользователь? Может ли пользователь работать со всеми явлениями одновременно или он ограничен только одним явлением? Дело в том, что у каждого явления есть определенный жизненный цикл (lifecycle). Явления располагаются в системе в специальном стеке, причем выполняемое в настоящий момент явление находится на самой верхней позиции в этом стеке. Если выполняемое явление запускает другое явление, то первое из этих явлений перемещается на самую нижнюю позицию в стеке, а новое размещается на самой верхней позиции. Явления, расположенные на более нижних позициях в стеке, могут быть приостановлены (paused) или остановлены (stopped). Приостановленное явление частично или полностью видно пользователю; остановленное

явление пользователю не видно. Система может завершать (kill) приостановленные или остановленные явления, если затрачиваемые на них ресурсы нужны для других целей.

Теперь подробнее поговорим о сохранении состояния данных (data persistence). Записи, создаваемые пользователем, сохраняются в базе данных на устройстве. В частности, вспомогательным запоминающим устройством (ЗУ) для приложения NotePad является база данных SQLite. Метод managedQuery(), рассмотренный нами ранее, при определенных условиях обращается к информации, содержащейся в базе данных, через поставщик содержимого. Проверим, как URI, переданный managedQuery(), выполняет запрос к базе данных SQLite. Напоминаем, что URI, передаваемый managedQuery(), имеет следующий вид:

```
public static final Uri CONTENT_URI =
Uri.parse("content://" + AUTHORITY + "/notes");
```

URI контента всегда имеет форму content://, за ней следует описание источника и общая часть (зависящая от контекста). Поскольку в URI не содержится самих данных, на основании такого URI система выполняет код, который, в свою очередь, производит данные. Какова эта связь? Как ссылка URI становится кодом, производящим данные? Является ли URI службой HTTP или веб-службой? Что касается самого URI или той его части, в которой содержится информация об источнике, то эта информация заключена в файле AndroidManifest.xml и имеет такую конфигурацию, как поставщик содержимого:

```
<provider android:name="NotePadProvider"
  android:authorities="com.google.provider.NotePad"/>
```

Когда Android сталкивается с URI, который необходимо разрешить (resolve), она извлекает фрагмент, описывающий источник и ищет класс ContentProvider, конфигурация которого соответствует указанному источнику. В приложении NotePad файл AndroidManifest.xml содержит класс NotePadProvider, конфигурация которого соответствует источнику com.google.provider.NotePad. В листинге 2.4 показан небольшой фрагмент этого класса.

Листинг 2.4. Класс NotePadProvider

```
public class NotePadProvider extends ContentProvider
{

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
  String[] selectionArgs,String sortOrder) {}

    @Override
    public Uri insert(Uri uri, ContentValues initialValues) {}

    @Override
    public int update(Uri uri, ContentValues values, String where,
  String[] whereArgs) {}

    @Override
```

```
    public int delete(Uri uri, String where, String[] whereArgs) {}

    @Override
    public String getType(Uri uri) {}

    @Override
    public boolean onCreate() {}

    private static class DatabaseHelper extends SQLiteOpenHelper {}

    @Override
       public void onCreate(SQLiteDatabase db) {}

       @Override
       public void onUpgrade(SQLiteDatabase db,
int oldVersion, int newVersion) {
          //...
       }
    }
}
```

Класс NotePadProvider дополняет класс ContentProvider. В классе ContentProvider определяется шесть абстрактных методов, четыре из которых образуют группу CRUD (Create, Read, Update, Delete), то есть выполняют операции создания, чтения, обновления и удаления данных. Еще два абстрактных метода — onCreate() и getType(). Метод onCreate() вызывается при первом создании поставщика содержимого. getType() предоставляет тип MIME для итогового набора данных (принцип действия типов MIME рассмотрен в главе 3).

Еще один интересный аспект, касающийся класса NotePadProvider, — это внутренний класс DatabaseHelper, дополняющий класс SQLiteOpenHelper. Вместе два этих класса отвечают за инициализацию базы данных NotePad, ее открытие и закрытие, а также за выполнение других задач, связанных с базой данных. Интересно отметить, что класс DatabaseHelper представляет собой всего несколько строк кода, добавляемого вручную (custom code) (листинг 2.5). Основную работу выполняет класс SQLiteOpenHelper, являющийся частью Android.

Листинг 2.5. Класс DatabaseHelper

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
       super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE " + NOTES_TABLE_NAME + " ("
             + Notes._ID + " INTEGER PRIMARY KEY,"
             + Notes.TITLE + " TEXT,"
             + Notes.NOTE + " TEXT,"
```

```
                + Notes.CREATED_DATE + " ,INTEGER,"
                + Notes.MODIFIED_DATE + " INTEGER"
                + ");");
    }

    //…
}
```

Из листинга 2.5 видно, что метод `onCreate()` создает таблицу NotePad. Обратите внимание — конструктор класса осуществляет вызов конструктора суперкласса, имеющего имя таблицы. Суперкласс вызывает метод `onCreate()` только в том случае, если таблица отсутствует в базе данных. Необходимо также отметить, что в таблице NotePad есть столбец `_ID`, рассмотренный нами выше.

Теперь опишем один из операторов CRUD: метод `insert()` (листинг 2.6).

Листинг 2.6. Метод insert()

```
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    long rowId = db.insert(NOTES_TABLE_NAME, Notes.NOTE, values);
    if (rowId > 0) {
        Uri noteUri = ContentUris.withAppendedId(
NotePad.Notes.CONTENT_URI, rowId);
        getContext().getContentResolver().notifyChange(noteUri, null);
        return noteUri;
    }
```

Метод `insert()` использует экземпляр внутреннего класса `DatabaseHelper` для доступа к базе данных, а затем вставляет запись. После этого возвращенный ID строки добавляется к URI, и новый URI возвращается к вызывающему оператору.

Теперь вам должна быть в целом понятна компоновка приложения в Android. Вы должны хорошо разбираться в программе NotePad и с некоторыми другими составляющими Android SDK. Вы должны уметь запускать образцы и экспериментировать с ними. В следующем разделе мы рассмотрим жизненный цикл приложения в Android.

Жизненный цикл приложения

Жизненный цикл приложения в Android жестко контролируется системой и зависит от нужд пользователя, доступных ресурсов и т. д. Например, пользователю может быть нужно запустить веб-браузер, но окончательное решение о том, запускать ли это приложение, принимает система. Хотя «последнее слово» и остается за системой, она подчиняется определенным заданным и логическим правилам, позволяющим определить, можно ли загрузить, приостановить приложение или прекратить его работу. Если в данный момент пользователь работает с определенным явлением, система дает приоритет соответствующему приложению. И наоборот, если явление невидимо и система решает, что работу приложения необходимо остановить, чтобы мобилизовать дополнительные ресурсы, будет прекращена работа приложения, имеющего более низкий приоритет.

Такой жизненный цикл отличается от жизненного цикла веб-приложений, создаваемых в J2EE. Приложения J2EE слабо управляются контейнером, в котором они работают. Например, контейнер J2EE может удалить приложение из оперативной памяти, если оно не проявляет активности в течение определенного периода времени. Но, как правило, контейнер не будет перемещать приложения в оперативную память и извлекать из нее по причинам, связанным с загруженностью системы и/или доступными ресурсами. Обычно контейнер J2EE располагает достаточным количеством ресурсов, чтобы обеспечивать одновременную работу нескольких приложений. Но в Android ресурсы более ограниченны, поэтому Android более жестко контролирует работу приложений.

ПРИМЕЧАНИЕ

В Android работа каждого приложения представляет собой отдельный процесс, выполняемый на специально выделенной для данного процесса виртуальной машине. Для этого необходима среда с защищенной памятью. Кроме того, при выполнении каждого приложения как отдельного процесса система может контролировать, какое приложение заслуживает более высокого приоритета. Например, фоновый процесс, выполняющий сложную вычислительную задачу, не может заблокировать входящий телефонный вызов.

Такая концепция жизненного цикла приложения логична, но из-за одного фундаментального свойства приложений Android возникает проблема. А именно — архитектура приложения Android является компонентно и интеграционно ориентированной. Благодаря этому достигается насыщенное взаимодействие системы с пользователем, органичное многократное применение приложения и легкая интеграция приложений. Но работа диспетчера, управляющего жизненным циклом приложений, при этом значительно осложняется.

Рассмотрим типичную ситуацию. Пользователь говорит с кем-либо по телефону, и вдруг ему требуется открыть электронное сообщение. Он переходит на рабочий стол, открывает электронное сообщение, щелкает на ссылке, содержащейся в письме, и отвечает на вопрос друга, прочитав на веб-странице данные о котировках акций. В таком сценарии используются четыре приложения: программа рабочего стола, программа для передачи речи, почтовый клиент и браузер. Когда пользователь переходит от одного приложения к другому, эти различные виды взаимодействия плавно перетекают один в другой. Однако в фоновом режиме система сохраняет и восстанавливает состояния приложений. Например, когда пользователь нажимает ссылку, содержащуюся в электронном сообщении, система сохраняет метаданные о выполняемом явлении, связанном с чтением электронного сообщения, и только после этого запускает явление браузера, который открывает URL. На самом деле система сохраняет данные о любом явлении, прежде чем перейти к другому явлению, поэтому есть возможность вернуться к выполнению предыдущего явления (например, если пользователь переходит в работе на шаг назад). Если возникнет дефицит памяти, системе придется остановить процесс, реализующий определенное явление, и возобновить его, когда это будет необходимо.

Android различает жизненный цикл приложения и его компонентов. Следовательно, вам нужно понимать события, связанные с жизненным циклом, и уметь ими управлять, чтобы создаваемые вами приложения были надежными и стабиль-

ными. Сначала познакомимся с различными методами обратного вызова жизненного цикла, используемыми с явлением (листинг 2.7).

Листинг 2.7. Методы жизненного цикла явления

```
protected void onCreate(Bundle savedInstanceState);
protected void onStart();

protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
```

В листинге 2.7 показан список методов жизненного цикла, вызываемых Android в ходе существования определенного явления. Чтобы создать стабильное приложение, важно понимать, когда именно каждый из методов вызывается системой. Обратите внимание — вам не нужно реагировать на все эти методы. Но если вы все же на них реагируете, не забывайте также вызывать версии суперклассов. На рис. 2.15 продемонстрированы переходы между состояниями.

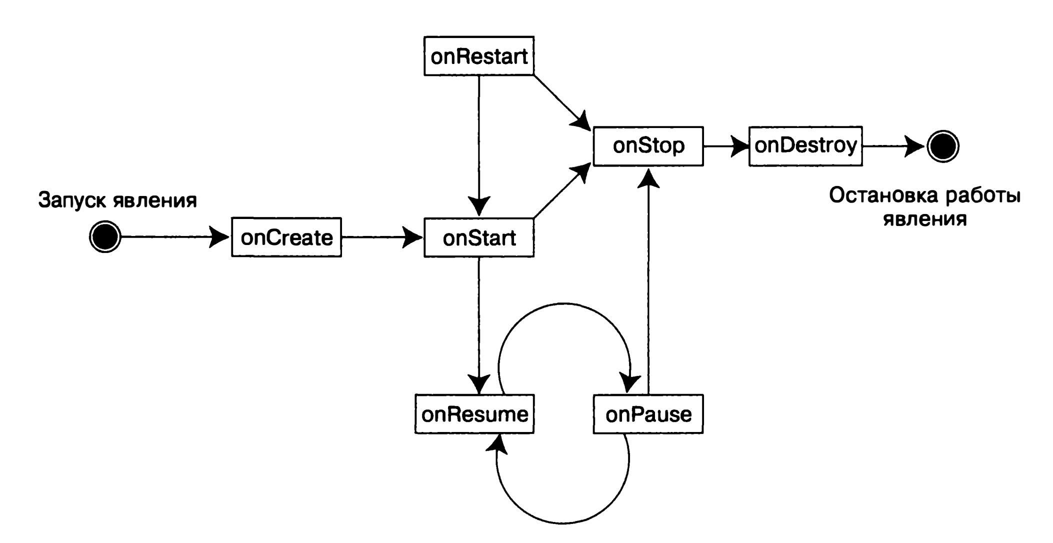


Рис. 2.15. Переходы между состояниями отдельно взятого явления

Система может запускать и останавливать текущие явления в зависимости от происходящих событий. Android вызывает метод `onCreate()`, когда явление только создается. За `onCreate()` всегда следует вызов `onStart()`, но перед `onStart()` не обязательно должен идти `onCreate()`, так как `onStart()` может вызываться и для возобновления работы приостановленного приложения (приложение останавливается методом `onStop()`). При вызове `onStart()` явление еще не видно пользователю, но вскоре будет видно. Метод `onResume()` вызывается после `onStart()`, даже когда явление работает в приоритетном режиме и пользователь может его наблюдать. В этот момент пользователь взаимодействует с созданным вами явлением.

Когда пользователь решает перейти к работе с новым явлением, система вызовет для прерываемого явления метод onPause(). От onPause() можно перейти к вызову либо onResume(), либо onStop(). Например, onResume() вызывается, когда пользователь возвращает конкретное явление в приоритетный режим работы. Метод onStop() вызывается, когда явление становится невидимым для пользователя. Если явление возвращается в приоритетный режим после вызова onStop(), при этом вызывается метод onRestart(). Если определенное явление находится в верхней позиции в стеке, но невидимо пользователю и система решает завершить это явление, вызывается метод onDestroy().

Описанная выше модель состояний явления может показаться сложной, но вам не обязательно применять все возможные сценарии. На практике вам чаще всего придется сталкиваться с onCreate() и onPause(). onCreate() будет вызываться при создании пользовательского интерфейса для работы с явлением. Данный метод позволит вам связывать данные с виджетами и подключать обработчики событий к компонентам пользовательского интерфейса. При помощи onPause() вы сможете сохранить важную информацию в базе данных вашего приложения. Это последний безопасный метод, который будет вызываться перед тем, как система завершит работу приложения. Методы onStop() и onDestroy() не обязательно будут вызываться, поэтому не полагайтесь на эти методы при реализации критической логики.

Какой же вывод следует из этого обсуждения? Система управляет приложением и может запускать, останавливать или возобновлять работу компонента программы в любое время. Хотя система и управляет компонентами, они работают не в полной изоляции, по крайней мере в рамках программы. Другими словами, если система запускает определенное явление в рамках вашей программы, вы можете положиться при выполнении явления на контекст приложения. Например, достаточно часто глобальные переменные совместно используются различными явлениями одного и того же приложения. Для предоставления глобальных переменных в совместное использование вы можете написать дополнение для класса android.app.Application, а затем инициализировать глобальную переменную методом onCreate() (листинг 2.8). После этого явления и другие компоненты вашей программы получат надежный доступ к этим ссылкам в ходе работы.

Листинг 2.8. Дополнение к классу приложения

```
public class MyApplication extends Application
{
   // глобальная переменная
   private static final String myGlobalVariable;

   @Override
   public void onCreate()
   {
      super.onCreate();
      //... здесь инициализируются глобальные переменные
      myGlobalVariable = loadCacheData();
   }

   public static String getMyGlobalVariable() {
```

```
        return myGlobalVariable;
    }
}
```

В следующем разделе рассмотрим еще один важный аспект создания приложений в Android — их отладку.

Отладка вашего приложения

Когда вы закончите несколько первых строк кода вашего первого приложения, возникнет вопрос: можно ли запустить сеанс отладки уже во время работы с приложением в эмуляторе. В Android SDK входит набор инструментов, предназначенных для отладки. Они интегрированы в Eclipse IDE (рис. 2.16).

Рис. 2.16. Инструменты отладки, используемые при написании приложений для Android

Один из инструментов, без которого не обойтись при разработке под Android, — это LogCat. Он отображает сообщения логов, рассылаемые при помощи `android.util.Log`, исключения, `System.out.println` и т. д. Пока `System.out.println` работает и в окне LogCat отображаются сообщения, вы можете использовать класс `android.util.Log` для регистрации сообщений программы. Этот класс определяет известные нам методы вывода информации, предупреждений и сообщений об ошибках. Указанные методы можно фильтровать в окне LogCat и просматривать только ту информацию, которая вас интересует. Команда Log может иметь следующий вид:

```
Log.v("string TAG", "Это мое сообщение для записи в логе");
```

В SDK также есть файловый менеджер (**File Explorer**), в котором файлы можно просматривать, перетаскивать в устройстве из каталога в каталог, даже если устройство на самом деле создано эмулятором.

Чтобы просмотреть эти инструменты, в Eclipse нужно выбрать режим **Debug** (Отладка). Можно также запустить каждый инструмент в режиме Java следующим образом: **Window ▸ Show View ▸ Other ▸ Android** (Окно ▸ Показать вид ▸ Другие ▸ Android).

Вы можете просмотреть подробные сведения о трассировке приложения Android, воспользовавшись классом `android.os.Debug`, в котором содержится метод для начала (`Debug.startMethodTracing()`) и для завершения трассировки (`Debug.stopMethodTracing()`). Android создаст в устройстве (или в эмуляторе) файл трассировки. Затем вы сможете скопировать файл трассировки на рабочую станцию и просмотреть вывод трассировщика при помощи специального инструмента, имеющегося в Android SDK. Другие инструменты будут подробно рассмотрены в следующих главах книги.

Резюме

В этой главе мы рассмотрели, как настроить среду разработки для создания приложений Android. Мы изучили некоторые базовые составляющие интерфейса прикладного программирования Android и познакомились с видами, явлениями, намерениями, поставщиками содержимого и службами. Затем мы разобрали приложение NotePad в контексте вышеназванных базовых элементов и компонентов приложения. Далее мы поговорили о важности жизненного цикла приложения в Android. И наконец, мы вкратце упомянули некоторые инструменты отладки, входящие в состав Android SDK и интегрируемые в Eclipse IDE.

С этого начинается наш путь разработки в Android. В следующей главе мы детально рассмотрим поставщики содержимого, ресурсы и намерения.

3 Использование ресурсов, поставщиков содержимого и намерений

В главе 2 мы в общем рассмотрели архитектуру приложения в Android и некоторые базовые концепции, лежащие в ее основе. Мы также изучили инструментарий для разработки в Android (SDK), Eclipse ADT (инструмент Eclipse для разработки в Android) и научились запускать приложения в эмуляторах, идентифицируемых при помощи виртуальных устройств Android (AVD).

В этой главе мы займемся углубленным изучением основ инструментария Android SDK и рассмотрим ресурсы, поставщики содержимого и намерения. Три этих элемента имеют определяющее значение для понимания программирования в Android и должны послужить вам базой для понимания материала, о котором пойдет речь в следующих главах.

Android использует ресурсы для описания компонентов пользовательского интерфейса в декларативной форме. Такой подход напоминает применение декларативных тегов в HTML для описания пользовательского интерфейса. В этом отношении разработка UI в Android является достаточно прямолинейной. В дальнейшем в Android такие ресурсы могут быть локализованы. В разделе «Ресурсы» мы рассмотрим различные виды ресурсов, применяемых в Android, и научимся ими пользоваться.

В Android используются так называемые *поставщики содержимого* (content providers) — сущности, предназначенные для обобщения данных в службах. Благодаря применению поставщиков содержимого источники данных становятся похожи на оснащенные функциями передачи состояния представления (REST) поставщики данных, например, на веб-сайты.

Веб-сайты отвечают за предоставление браузеру информации о типе данных, доступных по определенной URL, — точно так же поставщик содержимого обеспечивает описание возвращаемых им данных для каждой из предоставляемых служб. Подобно веб-сайтам, доступ к этим службам обработки данных осуществляется через унифицированные идентификаторы ресурсов (URI). В разделе «Поставщики содержимого» мы подробно объясним эту идею и покажем, как создать простой поставщик содержимого.

ПРИМЕЧАНИЕ

REST означает передачу состояния представления (REpresentational State Transfer). За этим не совсем понятным названием скрывается простая концепция, с которой знаком каждый пользователь Интернета. Когда вы вводите гиперссылку в адресную строку браузера и сервер присылает в ответ HTML-страницу, вы, по сути, осуществляете операцию, построенную на REST, — посылаете на сервер запрос (query). Подобным образом, когда вы обновляете определенный контент в веб-форме, обновление (update) на сервере (или на сайте) и соответствующее изменение состояния информации также происходит на основе REST. Обычно REST противопоставляется веб-службам SOAP (Simple Object Access Protocol, простой протокол доступа к объектам). Подробнее о REST можно прочитать в «Википедии»: http://en.wikipedia.org/wiki/Representational_State_Transfer.

В Android применяются ***намерения*** (intents), предназначенные для активирования компонентов пользовательского интерфейса (и компонентов вообще), а также для совместного использования данных между приложениями. В разделе, посвященном намерениям, мы рассмотрим, что это за сущность, и как намерения используются для обнаружения и активации программ пользовательского интерфейса, называемых ***явлениями*** (activities). Мы также изучим связи между намерениями, данными, унифицированными идентификаторами ресурсов и поставщиками содержимого. По ходу главы мы узнаем, как при помощи намерений обеспечивается гибкость и возможность многократного использования ресурсов в Android.

В целом, в этой главе сообщаются базовые знания, необходимые для дальнейшего освоения программирования в Android.

Ресурсы

Ресурсы — это основной компонент архитектуры Android. Из этого раздела вы узнаете, что ресурсы являются декларативными и что Android создает ID ресурсов для удобства работы с программами Java. Мы также увидим, как файл-источник `R.java` опосредует генерирование и использование таких ID. Затем мы изучим, как определять ресурсы в файлах XML, повторно использовать ресурсы в рамках другого объявления ресурса в XML и многократно использовать ресурсы в программах Java. Кроме таких ресурсов, основанных на XML, мы рассмотрим еще два типа ресурсов: ***исходные*** (raw) ***ресурсы*** и ***активы*** (assets).

Ресурс в Android представляет собой файл (например, музыкальный) или значение (например, заголовок или диалоговое окно), связанные с исполняемым приложением. Такие файлы и значения связаны с приложением таким образом, что их можно изменять без повторной компиляции или новой разработки приложения. Ресурсы участвуют в работе практически всех известных фреймворков пользовательских интерфейсов.

К известным типам ресурсов относятся строки (strings), цвета (colors) и точечные рисунки (bitmaps). В приложении не применяются жестко написанные строки кода — вместо них можно использовать, например, соответствующие идентификационные номера. Такая опосредованность позволяет изменять текст строкового ресурса, не изменяя исходного кода.

Давайте начнем обсуждение ресурсов со строки — ресурса одного из наиболее распространенных типов.

Строковые ресурсы

В Android можно определить несколько строковых ресурсов в одном или нескольких XML-файлах ресурсов. В таких XML-файлах содержатся определения строковых ресурсов, расположенные в подкаталоге /res/values. Имена XML-файлов выбираются произвольно, хотя, как правило, в качестве имени файла указывается strings.xml. В листинге 3.1 показан пример файла, в котором определяются строковые ресурсы.

Листинг 3.1. Пример файла strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">hello</string>
   <string name="app_name">hello appname</string>
</resources>
```

При создании или обновлении такого файла плагин Eclipse ADT автоматически создает или обновляет класс Java в базовом корневом пакете вашего приложения, который называется R.java, сообщая уникальные ID для двух определенных в файле строковых ресурсов.

Обратите внимание, где именно расположен файл R.java в следующем примере. Мы продемонстрировали структуру высокоуровневого каталога, входящего в состав проекта MyProject.

```
\MyProject
   \src
      \com\mycompany\android\my-root-package
      \com\mycompany\android\my-root-package\another-package
   \gen
      \com\mycompany\android\my-root-package\
      \com\mycompany\android\my-root-package\R.java
   \assets
   \res
   \AndroidManifest.xml
.....etc
```

ПРИМЕЧАНИЕ

Независимо от количества файлов ресурсов, в проекте содержится только один файл R.java.

При использовании файла строковых ресурсов, показанного в листинге 3.1, обновленный файл R.java будет содержать следующие записи:

```
public final class R {
   ...другие записи, зависящие от конкретного проекта и приложения

   public static final class string
   {
```

```
        ...другие записи, зависящие от конкретного проекта и приложения

            public static final int hello=0x7f040000;
            public static final int app_name=0x7f040001;
        ...другие записи, зависящие от конкретного проекта и приложения
    }
    ...другие записи, зависящие от конкретного проекта и приложения
}
```

Для начала рассмотрим, как именно файл R.java определяет класс верхнего уровня корневого пакета: public static final class R. В рамках этого внешнего класса R Android определяет внутренний класс — static final class string. R.java создает этот внутренний статический класс как пространство имен, в котором содержатся ID строковых ресурсов.

Два static final int, определяемых в переменных с именами hello и app_name, являются идентификаторами ресурсов, которые представляют соответствующие строковые ресурсы. Вы можете использовать эти идентификаторы в любой части исходного кода, соблюдая следующую структуру кода:

```
R.string.hello
```

Обратите внимание — сгенерированные ID указывают на int, а не на string. Большинство методов, использующих строки, также принимают эти идентификаторы в качестве ввода. Android при необходимости разрешает такие int в string.

Принято, что в примерах приложений все строковые ресурсы определяются в одном файле strings.xml. Android принимает любое количество произвольных файлов, если XML-файл имеет такую структуру, как в листинге 3.1, а сам файл находится в подкаталоге /res/values.

Структура этого файла довольно проста. Имеется корневой узел <resources>, за которым следуют один или несколько дочерних элементов <string>. Каждый элемент <string>, являющийся дочерним по отношению к <resources>, имеет свойство name, которое в файле R.java представляет собой атрибут id.

В этом подкаталоге могут содержаться и такие файлы, в которых определяется несколько строковых ресурсов. Давайте сохраним в том же подкаталоге еще один файл немного другого содержания и назовем его strings1.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello1">hello 1</string>
    <string name="app_name1">hello appname 1</string>
</resources>
```

Плагин Eclipse ADT проверяет уникальность этих ID во время компилирования и помещает их в файле R.java как две дополнительные константы: R.string.hello1 и R.string.app_name1.

Ресурсы разметки формы

В Android экранные виды часто представляют собой особые ресурсы и загружаются из файла XML. Ресурс разметки формы (layout resource) — это ключевой тип

ресурсов, применяемый при программировании пользовательских интерфейсов в Android. Следующий фрагмент кода является примером явления, используемого в Android:

```
public class HelloWorldActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {

      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      TextView tv = (TextView)this.findViewById(R.id.text1);
      tv.setText("Попробуйте этот текст");
   }
   .........
}
```

Строка setContentView(R.layout.main) указывает на то, что в примере имеется статический класс, называемый R.layout, а в этом классе есть константа main (натуральное число), указывающая на View, определяемый в XML-файле ресурса разметки формы. XML-файл будет иметь имя main.xml, он должен быть размещен в подкаталоге ресурсов layout. Иными словами, этот оператор предполагает, что программист создаст файл /res/layout/main.xml и поместит в нем необходимое определение разметки формы. Содержимое файла main.xml приведено в листинге 3.2.

Листинг 3.2. Пример файла разметки формы Example main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/hello"
   />
<Button android:id="@+id/b1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@+string/hello"
   />
</LinearLayout>
```

В файле разметки формы, приведенном в листинге 3.2, определяется корневой узел под названием LinearLayout. В нем содержится TextView, за которым следует Button. В узле LinearLayout задается шаблон дочерних элементов по горизонтали или вертикали (в данном случае по вертикали).

Для каждого экранного вида необходимо создавать отдельный файл разметки формы. Точнее говоря, для каждого варианта разметки требуется специальный файл. Если вы рисуете два экрана, вам понадобится два файла разметки, например /res/layout/screen1_layout.xml и /res/layout/screen2_layout.xml.

ПРИМЕЧАНИЕ

Каждый файл в подкаталоге /res/layout/ генерирует уникальную константу на основе имени файла (исключая расширение). При работе с ресурсами разметки здесь важно количество файлов, а при работе со строковыми ресурсами важно количество отдельных строковых ресурсов *внутри* файлов.

Например, если в подкаталоге /res/layout/ у нас есть два файла — file1.xml и file2.xml, в R.java будут содержаться следующие записи:

```
public static final class layout {
   .... все остальные файлы
      public static final int file1=0x7f030000;
      public static final int file2=0x7f030001;
}
```

Виды, определяемые в данных файла компоновки, в частности TextView (см. листинг 3.2), доступны в коде Java по их ID, генерируемым в R.java:

```
TextView tv = (TextView)this.findViewById(R.id.text1);
tv.setText("Попробуйте этот текст");
```

В данном примере мы находим файл TextView при помощи метода findViewById класса Activity. Константа R.id.text1 соответствует ID, заданному в TextView. Id для TextView в файле разметки выглядит следующим образом:

```
<TextView android:id="@+id/text1"
..
</TextView>
```

Значение атрибута id указывает, что константа, называемая text1, будет использоваться для уникальной идентификации этого вида среди других, используемых данным явлением. Знак «+» в @+id/text1 означает, что будет создан ID text1, если он еще не существует. Синтаксис такого ID ресурса достаточно сложен, мы подробно рассмотрим его в следующем разделе.

Синтаксис ссылок на ресурсы

Независимо от типа (пока мы рассмотрели два типа ресурсов — строковый ресурс (string) и ресурс разметки (layout)) все ресурсы Android идентифицируются по их id, содержащемуся в исходном коде Java (кроме того, на этот идентификатор ставятся ссылки). Синтаксис, используемый при соотнесении id с ресурсом в файле XML, называется *синтаксис ссылок на ресурсы* (resource-reference syntax). Синтаксис атрибута id в предыдущем примере @+id/text1 имеет следующую формальную структуру:

```
@[package:]type/name
```

Type соответствует одному из пространств имен, зависящих от типа ресурса и доступных в R.java. Некоторые из этих пространств имен перечислены ниже:

```
R.drawable
R.id
R.layout
R.string
R.attr
```

Соответствующие типы, используемые в синтаксисе ссылок на ресурсы в XML, таковы:

```
drawable
id
layout
string
attr
```

Часть name, содержащаяся в ссылке на ресурс @[package:]type/name, — это имя ресурса. Она также представлена в виде константы int в файле R.java.

Если в синтаксисе строки @[package:]type/name не указать пакет (package), то разрешение пары type/name будет производиться на основе локальных ресурсов и локального пакета R.java, используемого в программе.

Если указать android:type/name, разрешение ID ссылки будет производиться с применением пакета Android, в частности посредством файла android.R.java. Вы можете использовать имя любого пакета Java вместо подстановочного слова package, чтобы найти файл R.java, подходящий для разрешения ссылки. Проанализируем несколько конкретных примеров:

```
<TextView android:id="text">
// Ошибка компиляции, так как id не принимает
// необработанные текстовые строки.

<TextView android:id="@text">
// Неправильный синтаксис. Не хватает названия типа
// Вы получите сообщение об ошибке
// "No Resource type specified (не указан тип ресурса).

<TextView android:id="@id/text">
// Ошибка: не найдено ни одного ресурса, соответствующего id "text".
// Возможно, ранее вы не задали "text" как один из видов ID.

<TextView android:id="@android:id/text">
// Ошибка: "Ресурс не является общедоступным"
// означает, что такой id отсутствует в android.R.id.
// Чтобы такая запись была действительной, в файле Android R.java
// необходимо задать id с таким именем.

<TextView android:id="@+id/text">
// Успешно: создает id с названием "text" в файле R.java локального пакета.
```

Определение собственных идентификационных номеров ресурсов для последующего использования

Общий принцип присвоения id предполагает либо создание нового id для ресурса, либо использование одного из id, созданных в пакете Android. Однако id можно создавать и заранее, а потом использовать их в собственных пакетах.

Строка <TextView android:id="@+id/text"> в предыдущем фрагменте кода указывает, что id с названием text, будет использоваться в том случае, если этот id уже создан. Если id еще не существует, нужно создать новый идентификатор. В связи с этим возникает вопрос: может ли id, например, text уже существовать в файле R.java, чтобы такой идентификатор можно было использовать многократно?

Можно предположить, что такую задачу могла бы выполнять константа, например R.id.text, находящаяся в файле R.java, но R.java не поддается редактированию. Даже если бы было возможно внести такие изменения, файл приходилось бы заново генерировать после добавления, изменения или удаления любой информации из подкаталога /res/*.

Решение проблемы заключается в использовании тега ресурса под названием item для задания id, не связанного ни с каким конкретным ресурсом. Ниже приведен соответствующий пример:

```
<resources>
<item type="id" name="text"/>
</resources>
```

Здесь type описывает тип ресурса, в данном случае id. Когда id будет установлен, будет работать и следующее определение View:

```
<TextView android:id="@id/text">
..
</TextView>
```

Скомпилированные и нескомпилированные ресурсы Android

Итак, мы рассмотрели строковые ресурсы и ресурсы разметки. Мы также обсудили синтаксис ссылок на ресурсы, в том числе в контексте ресурсов разметки. Теперь опишем другой ключевой аспект ресурсов Android. Речь идет о том, что большинство ресурсов компилируются в двоичные файлы, перед тем как использоваться. Но это касается не всех ресурсов — некоторые используются без дополнительной обработки.

В Android поддержка ресурсов осуществляется преимущественно при помощи файлов двух типов — XML- и RAW-файлов (к последним относятся изображения, аудио и видео). Уже при работе с XML-файлами мы видели, что в большинстве случаев ресурсы определяются как значения внутри файла XML (это касается, например, строк), а иногда весь XML-файл является ресурсом (например, файл ресурса разметки).

Файлы, созданные в XML, также подразделяются на два типа: первые компилируются в двоичный формат, а вторые копируются на устройство «как есть». В примерах, рассмотренных нами выше, мы имели дело с XML-файлами строковых ресурсов и ресурсов разметки. Такие файлы компилируются в двоичный формат, прежде чем входят в состав инсталлируемого пакета. Эти XML-файлы имеют заданный формат, в котором узлы XML преобразуются в ID.

Вы также можете специально выбрать некоторые XML-файлы и задать для них структуру формата, отличающуюся от принятой по умолчанию, — чтобы эти файлы не интерпретировались и для них не генерировались идентификаторы ресурсов. Однако в данном случае нам как раз нужно, чтобы они компилировались в двоичный формат и чтобы их было удобно локализовывать. Для достижения этого файлы XML можно поместить в подкаталоге `/res/xml/` — тогда они будут скомпилированы в двоичный формат. В таком случае вы можете воспользоваться поставляемыми вместе с Android инструментами для считывания XML, которые могут считывать XML-узлы.

Но если разместить файлы, в том числе написанные на XML, в каталоге `/res/raw/`, они не будут скомпилированы в двоичном формате. Для считывания таких файлов нужно использовать явные API с поддержкой потоков. К категории raw относятся аудио- и видеофайлы.

ПРИМЕЧАНИЕ

Необходимо отметить, что, поскольку каталог raw относится к иерархии /res/*, даже необработанные аудио- и видеофайлы поддаются локализации, как и все другие ресурсы.

Как было указано в табл. 2.2, файлы ресурсов располагаются в различных подкаталогах в зависимости от типа файлов. Ниже перечислены некоторые важные подкаталоги, находящиеся в папке `/res`, и типы ресурсов, которые в них содержатся:

- `anim` — скомпилированные анимационные файлы;
- `drawable` — точечные рисунки;
- `layout` — определения компонентов пользовательского интерфейса/видов;
- `values` — массивы, цвета, параметры, строки и стили;
- `xml` — скомпилированные произвольные XML-файлы;
- `raw` — нескомпилированные необработанные (raw) файлы.

Компилятор ресурсов, входящий в состав инструмента Android Asset Packaging Tool (AAPT), собирает все ресурсы, кроме raw, и помещает все их в итоговый файл APK. Этот файл, содержащий код и ресурсы приложения Android, аналогичен файлу JAR, который применяется в Java (APK означает Android package — пакет Android). Именно файл APK устанавливается на устройство.

ПРИМЕЧАНИЕ

Хотя инструмент синтаксического разбора (parser) ресурсов XML допускает такие имена ресурсов, как hello-string, в файле R.java во время компиляции такое название вызовет ошибку. Чтобы ее устранить, можно переименовать ресурс в hello_string (заменив дефис нижним подчеркиванием).

Перечисление основных ресурсов Android

Итак, мы изучили основные концепции, касающиеся ресурсов. Теперь перечислим некоторые другие ключевые ресурсы, поддерживаемые в Android, их XML-представления и способ использования их в коде Java. (Этот раздел можно использовать как справочный при написании файлов для каждого конкретного типа ресурса.) Сначала рассмотрим типы ресурсов и узнаем, для чего используются те или иные ресурсы (табл. 3.1).

Таблица 3.1. Типы ресурсов

Тип ресурса	Размещение	Описание
Цвета	/res/values/any-file	Представляет собой идентификатор цвета, указывающий на цветовой код. ID таких ресурсов выражаются в R.java как R.color.*. Этот XML-узел в файле имеет вид /resources/color
Строки	/res/values/any-file	Представляет собой строковые ресурсы. В их число также входят строки в формате java и обычный html, а не только обычные строки. ID таких ресурсов выражаются в R.java как R.string.*. Этот XML-узел в файле имеет вид /resources/string
Параметры	/res/values/any-file	Представляет собой параметры или размеры различных элементов или видов в Android. Поддерживает пикселы, дюймы, миллиметры, не зависящие от плотности экрана пикселы (dip) и пикселы, не зависящие от масштаба. ID таких ресурсов выражаются в R.java как R.dimen.*. Этот XML-узел в файле имеет вид /resources/dimen
Изображения	/res/drawable/ multiple-files	Представляет ресурсы-изображения. Поддерживает форматы изображений JPG, GIF, PNG и др. Каждое изображение является отдельным файлом и получает собственный идентификатор, определяемый по имени файла. Такие ID ресурсов представляются в файле R.java как R.drawable.*. Система также поддерживает так называемые растягиваемые изображения (stretchable image), в которых можно менять масштаб отдельных элементов, а другие элементы оставлять без изменений
Отрисовываемые цвета	/res/values/any-file и /res/drawable/ multiple-files	Представляет цветные прямоугольники, которые используются в качестве фона основных отрисовываемых объектов, например точечных рисунков. В Java в таком случае создавался бы прямоугольник определенного цвета и устанавливался в качестве фона для вида (view). Поддержка такой функции обеспечивается тегом значения <drawable>, находящимся в подкаталоге значений. Такие id ресурсов выражаются в файле R.java как R.drawable.*. XML-узел для такого файла — /resources/drawable. В Android при помощи специальных XML-файлов, расположенных в /res/drawable, также поддерживаются скругленные и градиентные прямоугольники. Корневым XML-тегом для <drawable>является <shape>. Идентификаторы таких ресурсов также выражаются в файле R.java как R.drawable.*. В таком случае, каждое имя файла преобразуется в уникальный id отрисовываемого объекта
Произвольные XML-файлы	/res/xml/*.xml	В Android в качестве ресурсов могут использоваться произвольные XML-файлы. Они компилируются в aapt. Идентификаторы таких ресурсов также выражаются в файле R.java как R.xml.*

Тип ресурса	Размещение	Описание
Произвольные необработанные ресурсы	/res/raw/*.*	В этом каталоге в Android могут находиться нескомпилированные двоичные или текстовые файлы. Каждый файл получает уникальный id ресурса. Идентификаторы таких ресурсов также выражаются в файле R.java как R.raw.*
Произвольные необработанные активы	/assets/*.*/*.*	В Android могут присутствовать произвольные файлы, находящиеся в произвольно названных подкаталогах. Такие каталоги находятся в подкаталоге /assets. Это не ресурсы как таковые, а просто необработанные файлы. В этом каталоге, в отличие от /res, подкаталоги могут располагаться на любой глубине. Для таких файлов не создаются идентификаторы ресурсов. При работе с ними нужно использовать относительное имя пути, начиная с /assets, но не указывая этого каталога в имени пути

Все ресурсы, перечисленные в этой таблице, подробнее рассмотрены в следующих разделах и сопровождаются кодом на XML и Java.

ПРИМЕЧАНИЕ

Рассматривая, как именно генерируются id, может создаться впечатление (правда, нигде официально не подтвержденное), что id генерируются на основании имен файлов, если XML-файлы с такими именами находятся в любом подкаталоге, кроме res/values. Если XML-файлы располагаются в подкаталоге значений, при генерировании id учитывается только содержание этих файлов.

Цветовые ресурсы

Как и при работе со строковыми ресурсами, можно использовать идентификаторы ссылок (reference identifier) для опосредованного указания на цвета. Таким образом, в Android можно локализовывать цвета и применять темизацию. После того как вы определите и идентифицируете цвета в файлах ресурсов, в коде Java можно получить доступ к этим ресурсам по их ID. ID строковых ресурсов находятся в пространстве имен *<vash_paket>*`R.string`, а ID цветов — в пространстве имен *<vash_paket>*`.R.color`.

В Android также существуют отдельные файлы ресурсов, в которых определяется базовый набор цветов. Такие ID доступны в пространстве имен `android.R.color`. По следующей ссылке перечислены константные цветовые значения, доступные в пространстве имен `android.R.color`: **http://code.google.com/android/reference/android/R.color.html**.

В листинге 3.3 на примерах показано, как указываются цвета в файле ресурсов XML.

Листинг 3.3. Синтаксис XML для определения цветовых ресурсов

```
<resources>
  <color name="red">#f00</color>
  <color name="blue">#0000ff</color>
  <color name="green">#f0f0</color>
  <color name="main_back_ground_color">#ffffff00</color>
</resources>
```

Этот код должен находиться в файле, располагающемся в подкаталоге `/res/values`. Имя файла вы выбираете произвольно. Android прочтет все файлы, а затем

обработает их, а также найдет конкретные узлы, такие как «ресурсы» и «цвет», чтобы получить их ID.

В листинге 3.4 показано, как цветовые ресурсы используются в коде Java.

Листинг 3.4. Цветовые ресурсы в коде Java

```
int mainBackGroundColor
   = activity.getResources.getColor(R.color.main_back_ground_color);
```

В листинге 3.5 показано, как цветовые ресурсы должны использоваться при определении вида.

Листинг 3.5. Использование цветов при определении видов

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textColor="@color/ red"
          android:text="Образец текста для демонстрации красного цвета"/>
```

Подробнее о строковых ресурсах

Мы кратко рассмотрели строковые ресурсы в начале этой главы. Теперь изучим их более детально. Мы увидим, как определять и использовать строки, написанные на HTML, а также узнаем, как происходит подстановка переменных в строковых ресурсах.

ПРИМЕЧАНИЕ

В большинстве пользовательских интерфейсов, в которых применяются строковые ресурсы, Android позволяет быстро связывать ID со строковыми ресурсами в файле R.java. Таким образом, использование строк в качестве ресурсов Android значительно упрощается.

Начнем с того, как определять в XML-файлах ресурсов следующие виды строк: обычные строки, строки в кавычках, HTML-строки и заменяемые строки (листинг 3.6).

Листинг 3.6. Синтаксис XML при определении строковых ресурсов

```
<resources>
   <string name="simple_string">simple string</string>
   <string name="quoted_string">"quoted'string"</string>
   <string name="double_quoted_string">\"double quotes\"</string>
   <string name="java_format_string">
         hello %2$s java format string. %1$s again
   </string>
   <string name="tagged_string">
      Hello <b><i>Slanted Android</i></b>, You are bold.
   </string>
</resources>
```

Этот XML-файл строковых ресурсов должен находиться в подкаталоге /res/values. Имя файла выбирается произвольно.

Обратите внимание — строки, находящиеся в кавычках, необходимо либо пропускать, либо помещать еще в одни кавычки. При определении строк также можно использовать стандартные последовательности Java, предназначенные для форматирования строк.

В Android могут также применяться дочерние элементы XML, в частности <b>, <i> и другие простые элементы HTML, предназначенные для форматирования текста, — такие элементы следует ставить в узле <string>. Вы можете использовать такую составную HTML-строку для оформления текста, перед тем как рисовать текстовый вид.

Каждый вариант использования проиллюстрирован в листинге 3.7 на примере кода Java.

Листинг 3.7. Использование строковых ресурсов в коде Java

```
// Считывание обычной строки и помещение ее в текстовый вид.
String simpleString = activity.getString(R.string.simple_string);
textView.setText(simpleString);

// Считывание строки в кавычках и помещение ее в текстовый вид.
String quotedString = activity.getString(R.string.quoted_string);
textView.setText(quotedString);

// Считывание строки в двойных кавычках и помещение ее в текстовый вид.
String doubleQuotedString = activity.getString(R.string.double_quoted_string);
textView.setText(doubleQuotedString);

// Считывание строки форматирования Java.
String javaFormatString = activity.getString(R.string.java_format_string);
// Преобразование отформатированной строки при помощи данных,
// передаваемых в аргументах.
String substitutedString = String.format(javaFormatString,
        "Hello" , "Android");
// помещение вывода в текстовый вид
textView.setText(substitutedString);

// Считывание строки html string из ресурса и помещение ее в текстовый вид.
String htmlTaggedString = activity.getString(R.string.tagged_string);
// Преобразование строки во фрагмент текста,
// который может быть помещен в текстовом виде.
// Класс android.text.Html допускает рисование строк "html".
// Этот класс относится именно к Android,
// и в нем поддерживаются не все HTML-теги
Spanned textSpan = android.text.Html.fromHtml(htmlTaggedString);
// Поместить информацию в текстовый вид
textView.setText(textSpan);
```

После того как вы определите строки в виде ресурсов, вы можете вставить их прямо в вид, так, как вставляется TextView в определение шаблона этого TextView, заданное в XML. В листинге 3.8 показан пример, в котором строка на HTML помещена в TextView как текстовый контент.

Листинг 3.8. Использование строковых ресурсов в XML

```
<TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textAlign="center"
        android:text="@string/tagged_string"/>
```

TextView автоматически определяет, что эта строка написана на HTML, и соответствующим образом обрабатывает форматирование этой строки. Это очень красивый метод, так как вы можете быстро вставить в виды привлекательный текст, который будет входить в состав шаблона.

Ресурсы, связанные с параметрами

Пикселы, дюймы, точки — все это единицы измерения, которые могут входить в состав XML-шаблонов и кода Java. Эти ресурсы, связанные с параметрами, можно использовать для оформления и локализации пользовательских интерфейсов в Android, не изменяя исходного кода.

В листинге 3.9 показано, как в XML используются ресурсы, связанные с параметрами.

Листинг 3.9. Синтаксис XML для определения ресурсов, связанных с параметрами

```
<resources>
   <dimen name="mysize_in_pixels">1px</dimen>
   <dimen name="mysize_in_dp">5dp</dimen>
   <dimen name="medium_size">100sp</dimen>
</resources>
```

Параметры можно указывать в следующих единицах:

- *px* — пикселы (pixels);
- *in* — дюймы (inches);
- *mm* — миллиметры (millimeters);
- *pt* — точки (points);
- *dp* — независящие от плотности экрана (density-independent) пикселы на основе экрана с разрешением 160 dpi (количество пикселов на дюйм) (параметры корректируются в соответствии с растровой плотностью экрана);
- *sp* — пикселы, не зависящие от масштабирования (scale-independent). Такие параметры допускают настройку размеров, производимую пользователем; полезны при работе с шрифтами.

В Java вы должны иметь доступ к каждому экземпляру объекта Resources, чтобы найти значения его параметров. Это можно сделать, применив метод getResources к объекту activity (листинг 3.10). Когда у вас будет объект Resources, его можно запросить по id параметра, чтобы узнать значение этого параметра (см. листинг 3.10).

Листинг 3.10. Использование ресурсов параметров в коде Java

```
float dimen = activity.getResources().getDimension(R.dimen.mysize_in_pixels);
```

ПРИМЕЧАНИЕ

Метод Java call использует Dimension (целое слово), а в пространстве имен R.java для этого применяется сокращенная версия dimen.

Как и в Java, в XML при использовании ссылок на ресурсы, связанные с параметрами, применяется dimen, а не целое слово dimension (листинг 3.11).

Листинг 3.11. Использование ресурсов параметров в XML

```
<TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="@dimen/medium_size"/>
```

Ресурсы изображений

Android генерирует идентификаторы ресурсов для файлов изображений, расположенных в подкаталоге /res/drawable. Поддерживаются файлы изображений GIF, JPG и PNG. Для каждого файла изображения, который находится в этом каталоге, генерируется уникальный идентификатор, производный от имени файла. Например, если файл изображения называется sample_image.jpg, то для него будет сгенерирован идентификатор ресурса R.drawable.sample_image.

ВНИМАНИЕ

Если у вас будет два файла с именами, произведенными от одного и того же базового имени (base file name), возникнет ошибка. Кроме того, будут игнорироваться файлы, расположенные в подкаталогах /res/drawable. Любые расположенные здесь файлы считываться не будут.

Можно делать ссылки на изображения, находящиеся в /res/drawable определений других шаблонов XML, как это показано в листинге 3.12.

Листинг 3.12. Использование ресурсов-изображений в XML

```
<Button
     android:id="@+id/button1"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Dial"
     android:background="@drawable/sample_image"
/>
```

Вы также можете найти изображение при помощи средств программирования, используя Java, и сопоставить это изображение с определенным элементом пользовательского интерфейса, например кнопкой (листинг 3.13).

Листинг 3.13. Использование ресурсов-изображений в Java

```
// вызов getDrawable для получения изображения
BitmapDrawable d = activity.getResources().getDrawable(R.drawable.sample_image);

// Затем можно использовать отрисовываемый объект, чтобы установить фон
button.setBackgroundDrawable(d);

// или можно установить фон непосредственно по идентификатору ресурса.
button.setBackgroundResource(R.drawable.icon);
```

ПРИМЕЧАНИЕ

Такие методы работы с фоном восходят к классу View. В результате большинство элементов управления пользовательского интерфейса предоставляют такую поддержку фона.

В Android поддерживаются также особые *растягиваемые* изображения (stretchable image). К ним относятся файлы в формате PNG, некоторые части которых могут

быть статичными, а другие — растягиваемыми. В Android есть инструмент, называемый Draw 9-patch, при помощи которого на рисунке можно указывать такие области. Подробнее о нем написано по следующему адресу: **http://developer.android.com/guide/developing/tools/draw9patch.html**.

После того как будет доступно изображение PNG, вы можете работать с ним как с любым другим изображением. Такие изображения удобно использовать в качестве фона для кнопок, если кнопка должна растягиваться, чтобы правильно расположиться в тексте.

Ресурсы отрисовываемых цветов

В Android изображение является одним из типов отрисовываемых ресурсов. В Android поддерживается и другой тип отрисовываемых ресурсов, которые называются отрисовываемыми цветами (color-drawable). В принципе, это просто разноцветные прямоугольники.

ВНИМАНИЕ

Судя по документации Android, система должна поддерживать рисование закругленных углов. Но авторам не удалось создать такие углы. Мы расскажем об альтернативном подходе, позволяющем получить закругленные углы. Кроме того, исходя из документации инстанцированный класс Java должен называться PaintDrawable, но в коде присутствует ColorDrawable.

Чтобы задать такой цветной прямоугольник, элемент XML определяется по имени узла drawable, это касается любого XML-файла, находящегося в подкаталоге /res/values. В листинге 3.14 мы видим несколько примеров отрисовываемых цветов.

Листинг 3.14. Синтаксис XML для определения ресурсов типа отрисовываемый цвет

```
<resources>
   <drawable name="red_rectangle">#f00</drawable>
   <drawable name="blue_rectangle">#0000ff</drawable>
   <drawable name="green_rectangle">#f0f0</drawable>
</resources>
```

В листингах 3.15 и 3.16 показано, как использовать отрисовываемые цвета соответственно в Java и в XML.

Листинг 3.15. Использование отрисовываемых цветов в коде Java

```
// получение отрисовываемого объекта
ColorDrawble redDrawable =
(ColorDrawable)
activity.getResources().getDrawable(R.drawable.red_rectangle);

// установление его в качестве фона для текстового вида
textView.setBackground(redDrawable);
```

Листинг 3.16. Использование отрисовываемых цветов в коде XML

```
<TextView android:layout_width="fill_parent"
          android:layout_height="wrap_content"
          android:textAlign="center"
          android:background="@drawable/red_rectangle"/>
```

Чтобы получить в Drawable закругленные углы, можно использовать тег <shape>, который пока не описан в документации. Однако этот тег должен находиться в файле, который расположен в каталоге /res/drawable. В листинге 3.17 показано, как использовать тег <shape> для определения скругленного прямоугольника в файле /res/drawable/my_rounded_rectangle.xml.

Листинг 3.17. Определение скругленного прямоугольника

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
   <solid android:color="#f0600000"/>
   <stroke android:width="3dp" color="#ffff8080"/>
   <corners android:radius="13dp" />
   <padding android:left="10dp" android:top="10dp"
      android:right="10dp" android:bottom="10dp" />
</shape>
```

Вы можете использовать этот отрисовываемый ресурс в качестве фона в предыдущем примере, где мы работали с текстовым видом:

```
// получение отрисовываемого объекта
GradientDrawable roundedRectangle =
(GradientDrawable)
activity.getResources().getDrawable(R.drawable.red_rectnagle);

// установление этого объекта в качестве фона для текстового вида
textView.setBackground(roundedRectangle);
```

ПРИМЕЧАНИЕ

Можно не приводить возвращенный базовый объект Drawable к виду GradientDrawable, но мы сделали это в примере, чтобы продемонстрировать, как тег <shape> становится GradientDrawable. Это важная информация, поэтому можете подробнее почитать об этом классе в документации по API Java, чтобы знать, какие XML-теги в нем определяются.

В итоге точечное изображение в подкаталоге отрисовываемых объектов разрешается в класс BitmapDrawable. Значение drawable, которое имеет ресурс, например один из показанных выше прямоугольников, разрешается в ColorDrawable. XML-файл, содержащий тег shape, разрешается в GradientDrawable.

Работа с произвольными XML-файлами ресурсов

В Android в качестве ресурсов могут также использоваться произвольные XML-файлы. Такой подход распространяет три обычных достоинства «ресурсов» на произвольные XML-файлы. Во-первых, появляется возможность быстрого проставления ссылок на эти файлы, так как для них генерируются ресурсные идентификаторы. Во-вторых, вы можете локализовывать эти XML-файлы ресурсов. В-третьих, вы можете организовать эффективное компилирование и хранение этих XML-файлов на устройстве.

XML-файлы, которые должны считываться таким образом, сохраняются в подкаталоге /res/xml. Ниже приведен пример XML-файла, который называется /res/xml/test.xml:

```
<rootelem1>
   <subelem1>
```

```
      Hello World from an xml sub element
   </subelem1>
</rootelem1>
```

Как и при работе с другими XML-файлами ресурсов, AAPT компилирует такой XML-файл, перед тем как поместить его в пакет прикладных программ. Для синтаксического разбора подобных файлов используйте экземпляр XmlPullParser. Чтобы реализовать в программе экземпляр XmlPullParser, применяйте следующий код в любом контексте (в том числе с activity):

```
Resources res = activity.getResources();
XmlResourceParser xpp = res.getXml(R.xml.test);
```

Возвращенный XmlResourceParser является экземпляром XmlPullParser, а также реализует java.util.AttributeSet. В листинге 3.18 показан более крупный фрагмент кода, считывающего файл test.xml.

Листинг 3.18. Использование XmlPullParser

```
private String getEventsFromAnXMLFile(Activity activity)
throws XmlPullParserException, IOException
{
   StringBuffer sb = new StringBuffer();
   Resources res = activity.getResources();
   XmlResourceParser xpp = res.getXml(R.xml.test);

   xpp.next();
   int eventType = xpp.getEventType();
      while (eventType != XmlPullParser.END_DOCUMENT)
      {
         if(eventType == XmlPullParser.START_DOCUMENT)
         {
            sb.append("******Start document");
         }
         else if(eventType == XmlPullParser.START_TAG)
         {
            sb.append("\nStart tag "+xpp.getName());
         }
         else if(eventType == XmlPullParser.END_TAG)
         {
            sb.append("\nEnd tag "+xpp.getName());
         }
         else if(eventType == XmlPullParser.TEXT)
         {
            sb.append(„\nText „+xpp.getText());
         }
         eventType = xpp.next();
      }// eof-while
      sb.append("\n******End document");
      return sb.toString();
}//eof-function
```

В листинге 3.18 показано, как получить XmlPullParser, как применять его для перехода между XML-элементами в XML-документе, и как пользоваться дополнительными методами XmlPullParser для доступа к подробной информации об XML-элементах. Если вы хотите испробовать этот код, то вам следует создать XML-файл так, как это показано выше, и вызвать функцию getEventsFromAnXMLFile из любого элемента меню либо нажать кнопку. Будет возвращена строка, которую вы сможете распечатать в потоке записей (log stream) при помощи метода отладки Log.d.

Использование необработанных ресурсов

Дополнительно к XML-файлам в Android можно использовать необработанные (raw) файлы. Такие необработанные ресурсы размещаются в /res/raw и представляют собой аудио-, видео- или текстовые файлы, для работы с которыми требуются локализация или ссылки, выставляемые на идентификаторы ресурсов. В отличие от XML-файлов, размещаемых в /res/xml, необработанные файлы не компилируются, а попадают в пакет прикладных программ как есть. Но каждый такой файл имеет идентификатор, генерируемый в R.java. Если бы вы поместили текстовый файл в /res/raw/test.txt, то его можно было бы прочитать при помощи кода, приведенного в листинге 3.19.

Листинг 3.19. Считывание необработанного ресурса

```
String getStringFromRawFile(Activity activity)
{
      Resources r = activity.getResources();
      InputStream is = r.openRawResource(R.raw.test);
      String myText = convertStreamToString(is);
      is.close();
      return myText;
}

String convertStreamToString(InputStream is)
{
      ByteArrayOutputStream baos = new ByteArrayOutputStream();
      int i = is.read();
      while (i != -1)
      {
         baos.write(i);
         i = baos.read();
      }
      return baos.toString();
}
```

ВНИМАНИЕ

Имена файлов, в которых дублируются базовые имена, вызывают в плагине Eclipse ADT ошибку компиляции (build error). Так происходит со всеми идентификаторами ресурсов, которые сгенерированы для ресурсов, созданных на основе файлов.

Работа с активами

В Android имеется еще один каталог, в котором могут храниться файлы, предназначенные для включения в пакет /assets. Этот каталог находится на том же уровне, что и /res, то есть не является подкаталогом /res. Для файлов, располагающихся в /assets, в R.java не генерируются идентификаторы ресурсов. Для их считывания необходимо указать путь к файлу. Путь к файлу является относительным и начинается с /assets. Для доступа к этим файлам используется класс AssetManager:

```
// Примечание: в данном коде не показаны исключения.
String getStringFromAssetFile(Activity activity)
{
 AssetManager am = activity.getAssets();
 InputStream is = am.open("test.txt");
 String s = convertStreamToString(is);
 is.close();
 return s;
}
```

Просмотр структуры каталогов с ресурсами

Итак, рассмотрим, как выглядит общая структура каталогов, содержащих ресурсы:

```
/res/values/strings.xml
        /colors.xml
        /dimens.xml
        /attrs.xml
        /styles.xml
   /drawable/*.png
        /*.jpg
        /*.gif
        /*.9.png
   /anim/*.xml
   /layout/*.xml
   /raw/*.*
   /xml/*.xml
/assets/*.*/*.*
```

ПРИМЕЧАНИЕ

Поскольку каталог /assets находится на одном уровне с /res, только в /assets может располагаться любой набор подкаталогов. Файлы, находящиеся в любом другом каталоге, размещаются именно на уровне этого каталога и не глубже. Такая организация требуется, чтобы в R.java могли генерироваться идентификаторы для этих файлов.

Завершая этот раздел, обобщим уже изученный материал о ресурсах. Вы уже знаете, какие типы ресурсов поддерживаются в Android, и умеете создавать такие ресурсы в XML-файлах. Вы знаете, как генерируются идентификаторы ресурсов и как использовать их в коде Java. Вы также изучили, что генерирование ресурсов —

это удобный механизм, упрощающий работу с ресурсами в Android. Наконец, вы научились обращаться с необработанными ресурсами и активами. Теперь займемся поставщиками содержимого и научимся работать с данными в Android.

Поставщики содержимого

Поставщик содержимого (content provider) — это оболочка (wrapper), в которую заключены данные. В Android существует возможность выражения источников данных (или поставщиков данных) при помощи передачи состояния представления — REST, в виде абстракций, называемых *поставщиками содержимого*. База данных SQLite, используемая в устройстве Android, представляет собой такой источник данных, который можно заключить в поставщик содержимого. Чтобы получить данные из поставщика содержимого или сохранить в нем новую информацию, нужно использовать набор REST-подобных идентификаторов URI. Например, если бы вам было нужно получить набор книг из поставщика содержимого, в котором заключена электронная библиотека, вам понадобился бы такой URI:

```
content://com.android.book.BookProvider/books
```

Чтобы получить из библиотеки конкретную книгу (например, книгу № 23), будет использоваться следующий URI:

```
content://com.android.book.BookProvider/books/23
```

В этом разделе мы рассмотрим, как такие URI преобразуются в механизмы, обеспечивающие доступ к базе данных. Любая программа, работающая в устройстве, может использовать такие URI для доступа к данным и осуществления с ними определенных операций. Следовательно, поставщики содержимого играют важную роль при совместном использовании данных несколькими приложениями.

Однако функции поставщика содержимого гораздо теснее связаны с инкапсуляцией данных, чем с обеспечением доступа к ним. Для доступа к данным следует использовать механизм, предназначенный специально для этого, например базу данных SQLite, либо получать данные из базовых источников путем доступа через сеть. Итак, поставщик содержимого применяется лишь в тех случаях, когда вы хотите использовать данные совместно с внешним приложением либо в нескольких приложениях, работающих в устройстве. Для внутрисистемного доступа к данным программа может использовать любой подходящий механизм хранения данных (или доступа к ним), например один из следующих механизмов:

- *настройки* — наборы пар «ключ/значение», в которых можно сохранять настройки программы;
- *файлы* — файлы, которые являются по отношению к программе внешними и которые можно хранить на съемном носителе;
- *SQLite* — базы данных SQLite; доступ к каждой из них открыт только из того пакета, в котором была создана эта база данных;
- *сеть* — механизм, позволяющий удаленно получать или сохранять данные через Интернет.

ПРИМЕЧАНИЕ

Несмотря на то что в Android применяется целый ряд механизмов для доступа к данным, в этой главе мы сосредоточимся на изучении SQLite и поставщиков содержимого. Поскольку именно на базе поставщиков содержимого осуществляется совместное использование данных, которое применяется в Android значительно активнее, чем в других фреймворках с пользовательскими интерфейсами. Работа с данными через сеть будет рассмотрена в главе 8, а управление настройками приложения — в главе 11.

В этом разделе мы рассмотрим, какие поставщики содержимого используются в Android. Мы подробно обсудим структуру уникальных идентификаторов контента и узнаем, как эти идентификаторы связываются с типами MIME. После того как мы детально рассмотрим поставщики содержимого, мы изучим, как написать поставщик содержимого с нуля и вложить в него простую электронную библиотеку.

Исследование встроенных поставщиков в Android

В Android используются встроенные поставщики содержимого, документированные в пакете `android.provider`, который посвящен Java и расположен в инструментарии разработчика. По следующей ссылке приводится список поставщиков содержимого, используемых в Android: **http://developer.android.com//reference/android/provider/package-summary.html**.

Вот неполный список поставщиков содержимого, рассмотренных в файле документации:

```
Browser
CallLog
Contacts
   People
   Phones
   Photos
   Groups
MediaStore
   Audio
      Albums
      Artists
      Genres
      Playlists
   Images
      Thumbnails
   Video
Settings
```

ПРИМЕЧАНИЕ

Список поставщиков содержимого может немного варьироваться в зависимости от используемой версии Android. Этот список приведен для того, чтобы вы могли составить представление об имеющихся поставщиках содержимого, и не является универсальным.

На верхних уровнях иерархии располагаются базы данных, на нижних — таблицы. Так, `Browser`, `CallLog`, `Contacts`, `MediaStore` и `Settings` — это отдельные базы данных

SQLite, инкапсулированные в форме поставщиков. Обычно такие базы данных SQLite имеют расширение DB и доступ к ним открыт только из пакетов реализации (implementation package). Любой доступ к базе данных из-за пределов этого пакета осуществляется через интерфейс поставщика содержимого.

Исследование баз данных на эмуляторе и доступных устройствах

Поскольку многие поставщики содержимого в Android используют базы данных SQLite (http://www.sqlite.org/), в вашем распоряжении при испытании баз данных будут инструменты, предоставляемые как в Android, так и в SQLite. Многие из этих инструментов расположены в подкаталоге `\android-sdk-installdirectory\tools`.

ПРИМЕЧАНИЕ

В главе 2 было описано, как найти каталог tools и активировать командное окно в различных операционных системах. В этой главе, как и в большинстве остальных, даются примеры, касающиеся в основном платформ Windows. В данном разделе мы рассмотрим программы, предназначенные для работы с командной строкой, поэтому особое внимание будет уделяться названиям исполняемых и пакетных файлов, а не тому, в каких каталогах они расположены. В главе 2 мы рассмотрели, как задавать путь к каталогу с инструментами на различных платформах.

Один из инструментов — это удаленная оболочка (remote shell). Она расположена в устройстве, с которого вы можете работать с инструментом командной строки SQLite, открывающим доступ к указанной базе данных. В этом разделе мы рассмотрим, как использовать такую утилиту командной строки при испытании баз данных, интегрированных в Android.

В Android используется и другой инструмент, работающий с командной строкой, — `Android Debug Bridge (adb)`, который доступен в `tools\adb.exe`.

`adb` — это особый компонент инструментария Android, через который большинство других инструментов получают доступ к устройству. Однако для работы с ним вам будут нужны действующий эмулятор либо устройство Android, подключенное к `adb`. Чтобы узнать, есть ли в вашем распоряжении работающие устройства или эмуляторы, введите в командную строку такую последовательность:

```
adb devices
```

Если эмулятор не работает, вы можете запустить его, введя в командную строку следующее:

```
\tools\emulator.exe @avdname
```

Аргумент `@avdname` — это имя AVD (виртуального устройства Android). Виртуальные устройства Android и порядок их создания были рассмотрены в главе 2. Чтобы узнать, какие виртуальные устройства у вас уже есть и могут быть запущены, выполните следующую команду:

```
\tools\android list   avd
```

Эта команда выводит список доступных AVD. Если вы уже разрабатывали какие-либо программы для Android и запускали их в Eclipse ADT, то обязательно должны были сконфигурировать хотя бы одно виртуальное устройство. Поэтому

команда, указанная выше, выведет название как минимум одного виртуального устройства.

Ниже приведен пример вывода такой команды. (В зависимости от того, где находится ваш каталог с инструментами, а также от версии Android, следующий код может иметь разные номера версии, а также может отличаться путь к нему, например i:\android.)

```
I:\android\tools>android list avd
Available Android Virtual Devices:
      Name: avd
      Path: I:\android\tools\..\avds\avd3
      Target:  Google APIs (Google Inc.)
            Based on Android 1.5 (API level 3)
      Skin: HVGA
      Sdcard:  32M
---------
      Name: titanium
      Path: C:\Documents and Settings\Satya\.android\avd\titanium.avd
      Target:  Android 1.5 (API level 3)
      Skin: HVGA
```

AVD подробно рассматриваются в главе 2.

Вы также можете запустить эмулятор при помощи плагина ADT Eclipse. Это происходит автоматически, когда вы выбираете в эмуляторе программы для запуска или отладки. После того как эмулятор будет запущен, вы снова сможете проверить список работающих устройств, введя следующую команду:

```
\tools\adb.exe devices
```

Вы должны увидеть следующий вывод:

```
List of devices attached
emulator-5554 device
```

Чтобы увидеть весь список параметров и команд, которые можно запустить при помощи adb, введите в командной строке следующее:

```
adb help
```

Кроме того, рекомендуем посетить следующий сайт, где перечислены многие параметры времени исполнения (runtime options) для adb: **http://developer.android.com/guide/developing/tools/adb.html**.

Чтобы запустить оболочку (shell) на подключенном устройстве, выполните следующую команду:

```
\tools\adb.exe shell
```

ПРИМЕЧАНИЕ

В принципе, эта оболочка аналогична ash, применяемой в Unix, но имеет ограниченный набор команд. Например, здесь можно выполнить ls, но find, grep и awk в этой оболочке недоступны.

Вы можете просмотреть список команд, доступных при работе в оболочке, введя следующую информацию по приглашению оболочки:

```
#ls   /system/bin
```

Символ # обозначает приглашение оболочки (shell prompt). Для краткости в некоторых из следующих примеров мы пропустим такое приглашение. Предыдущая строка загружает команды, перечисленные в табл. 3.2. (Обратите внимание на то, что эти команды приведены только в качестве примера, а не как подробный список. Этот список может отличаться в зависимости от того, с какой именно версией Android вы работаете.)

Таблица 3.2. Список команд, доступных при работе с оболочкой

dumpcrash	sh	date
am	hciattach	dd
dumpstate	sdptool	cmp
input	logcat	cat
itr	servicemanager	dmesg
monkey	dbus-daemon	df
pm	debug_tool	getevent
svc	flash_image	getprop
ssltest	installd	hd
debuggerd	dvz	id
dhcpcd	hostapd	ifconfig
hostapd_cli	htclogkernel	insmod
fillup	mountd	ioctl
linker	qemud	kill
logwrapper	radiooptions	ln
telnetd	toolbox	log
iftop	hcid	lsmod
mkdosfs	route	ls
mount	setprop	mkdir
mv	sleep	dumpsys
notify	setconsole	service
netstat	smd	playmp3
printenv	stop	sdutil
reboot	top	rild
ps	start	dalvikvm
renice	umount	dexopt
rm	vmstat	surfaceflinger
rmdir	wipe	app_process
rmmod	watchprops	mediaserver
sendevent	sync	system_server
schedtop	netcfg	
ping	Chmod	

Чтобы просмотреть список каталогов и файлов, находящихся на корневом уровне (root-level), введите в командной строке оболочки следующую команду:

```
ls   -l
```

Чтобы просмотреть список баз данных, необходимо попасть в следующий каталог:

```
ls   /data/data
```

В этом каталоге содержится список пакетов, установленных на устройстве. Для примера исследуем пакет com.android.providers.contacts:

```
ls   /data/data/com.android.providers.contacts/databases
```

Эта команда отобразит файл contacts.db, представляющий собой базу данных SQLite.

ПРИМЕЧАНИЕ

Необходимо отметить, что в Android базы данных могут создаваться при первом обращении к ним. Это означает, что вы можете не увидеть данного файла, если еще не обращались к приложению contacts.

Если бы в оболочке ash была команда find, вы могли бы просмотреть все файлы базы данных *.db. Но не стоит производить эту операцию, имея в распоряжении только команду ls. Лучше всего поступить так:

```
ls -R /data/data/*/databases
```

Работая с этой командой, вы увидите, что в дистрибутиве Android имеются следующие базы данных (опять-таки в зависимости от версии Android набор может варьироваться):

```
alarms.db
contacts.db
downloads.db
internal.db
settings.db
mmssms.db
telephony.db
```

Можно активировать команду sqlite3 для одной из перечисленных баз данных, введя следующую информацию:

```
#sqlite3  /data/data/com.android.providers.contacts/databases/contacts.db
```

Для завершения работы с sqlite3 напишите:

```
sqlite>.exit
```

Обратите внимание: приглашение для adb — это #, а приглашение для sqlite3 — это sqlite>. О различных командах sqlite3 рассказано на следующем сайте: **http://www.sqlite.org/sqlite.html**. Однако ниже мы перечислим некоторые важнейшие команды, чтобы вам не приходилось искать их в Интернете. Чтобы просмотреть список таблиц, введите:

```
sqlite> .tables
```

Эта команда предоставляет быстрый доступ к:

```
SELECT name FROM sqlite_master
WHERE type IN ('table','view') AND name NOT LIKE 'sqlite_%'
UNION ALL
SELECT name FROM sqlite_temp_master
WHERE type IN ('table','view')
ORDER BY 1
```

Как вы уже, наверное, догадались, таблица sqlite_master — это главная таблица (master table), в которой отслеживаются таблицы и виды, содержащиеся в базе данных. Следующая команда распечатывает инструкцию create для таблицы, называемой people и находящейся в базе данных contacts.db:

```
.schema people
```

Это один из способов, позволяющих узнать названия всех столбцов, которые содержатся в таблице базы данных. При работе с поставщиками содержимого вы подробнее познакомитесь с этими столбцами, поскольку от них зависят методы доступа.

Однако было бы очень утомительно вручную делать синтаксический разбор длинной инструкции create — и только для того, чтобы изучить названия столбцов таблицы и их типы. К счастью, есть обходной метод: можно переместить contacts.db в самый низ окна на локальном компьютере, а затем проверить базу данных при помощи любого инструмента с графическим пользовательским интерфейсом — подобные инструменты были разработаны для версии SQLite 3. Чтобы переместить вниз файл contacts.db, можно дать следующую команду по приглашению операционной системы:

```
adb pull /data/data/com.android.providers.contacts/databases/contacts.db ➥
c:/somelocaldir/contacts.db
```

Мы воспользовались бесплатной программой Sqliteman (**http://sqliteman.com/**) — удобным и практичным инструментом с графическим пользовательским интерфейсом, предназначенным для работы с базами данных SQLite. Не все получилось сразу, но в целом авторы нáшли эту программу полностью подходящей для исследования баз данных, применяемых в Android.

Азбука SQLite

На следующих примерах команд SQL вы быстро научитесь ориентироваться в базе данных SQLite:

```
// установить заголовки столбцов, которые будут отображаться в программе
sqlite>.headers on

// выделить все строки таблицы
select * from table1;

// вычислить количество строк в таблице
select count(*) from table1;

// выбрать определенный набор столбцов
```

```
select col1, col2 from table1;

// выделить различные значения в столбце
select distinct col1 from table1;

// вычислить различные значения
select count(col1) from (select distinct col1 from table1);

// сгруппировать по
select count(*), col1 from table1 group by col1;

// регулярное внутреннее объединение
select * from table1 t1, table2 t2
where t1.col1 = t2.col1;

// оставшееся внешнее соединение
// дать всю информацию в t1, даже если в t2 нет строк
select * from table t1 left outer join table2 t2
on t1.col1 = t2.col1
where ....
```

Архитектура поставщиков содержимого

Вы уже изучили, как исследовать имеющиеся поставщики содержимого при помощи инструментов Android и SQLite. Далее мы познакомимся с некоторыми элементами поставщиков содержимого и узнаем, как они соотносятся с другими абстракциями, обеспечивающими доступ к данным.

Явления, аналогичные поставщикам содержимого, встречаются в таких областях компьютерной индустрии, как:

- веб-сайты;
- REST (передача состояния представления);
- веб-службы;
- хранимые процедуры.

Сначала рассмотрим, в чем заключается сходство содержимого поставщиков и веб-сайтов. Каждый поставщик содержимого, расположенный в устройстве, регистрируется как веб-сайт, при помощи специальной строки (она похожа на доменное имя, но называется *источником* (authority)). Такая уникальная в пределах устройства последовательность символов представляет собой основу для набора URI, которые может предложить поставщик содержимого. Этот механизм напоминает работу веб-сайта, который обладает доменным именем, но может содержать ряд гиперссылок (URL), по которым доступны лежащие на сайте документы и контент вообще.

Регистрация источника осуществляется в файле AndroidManifest.xml. Ниже приведено два примера, в которых показано, как можно регистрировать поставщики содержимого в AndroidManifest.xml:

```
<provider android:name="SomeProvider"
   android:authorities="com.your-company.SomeProvider" />

<provider android:name="NotePadProvider"
   android:authorities="com.google.provider.NotePad"
/>
```

Источник поставщика содержимого аналогичен доменному имени сайта. Если источник уже зарегистрирован, эти поставщики содержимого будут представлены гиперссылками, начинающимися с соответствующего префикса источника:

```
content://com.your-company.SomeProvider/
content://com.google.provider.NotePad/
```

Итак, поставщики содержимого, как и веб-сайты, имеют базовое доменное имя, действующее как стартовая URL.

ПРИМЕЧАНИЕ

Необходимо отметить, что поставщики содержимого, используемые в Android, могут иметь не совсем полное имя источника. В настоящее время полное имя источника рекомендуется использовать только со сторонними поставщиками содержимого. Поэтому вам иногда могут встретиться поставщики содержимого, состоящие из одного слова, например contacts, в то время как полное имя такого поставщика содержимого — com.google.android.contacts (такое имя использовалось бы, если бы это был сторонний поставщик содержимого).

В поставщиках содержимого также встречаются REST-подобные гиперссылки, предназначенные для поиска данных и работы с ними. В случае описанной выше регистрации унифицированный идентификатор ресурса, предназначенный для обозначения каталога или коллекции записей в базе данных NotePadProvider, будет иметь имя:

```
content://com.google.provider.NotePad/Notes
```

URI для идентификации отдельно взятой записи будет иметь вид:

```
content://com.google.provider.NotePad/Notes/#
```

где # соответствует конкретной записи. Ниже приведено еще несколько примеров URI, которые могут присутствовать в поставщиках содержимого:

```
content://media/internal/images
content://media/external/images
content://contacts/people/
content://contacts/people/23
```

Обратите внимание — здесь поставщики содержимого "media" (content://media) и "contacts" (content://contacts) имеют неполную структуру. Это обусловлено тем, что данные поставщики содержимого не являются сторонними и контролируются Android.

Поставщики содержимого также обладают некоторыми характеристиками сетевых служб. Через свои URI поставщик содержимого предоставляет внутренние данные в виде служб. Однако конечной информацией, получаемой по URL поставщика

содержимого, являются нетипизированные данные (not typed data), которые предоставляются при вызове веб-службы, основанной на простом протоколе доступа к объектам (SOAP). Такой вывод больше похож на результат запроса, возвращаемый оператором JDBC. Сходство с JDBC является концептуальным. Но между поставщиком содержимого и ResultSet нельзя поставить знак равенства.

Вызывающий оператор должен знать структуру возвращаемых строк и столбцов. Кроме того, как вы увидите в разделе «Структурирование MIME-типов в Android» этой главы, в поставщик содержимого встроен механизм, позволяющий определять тип MIME (Multipurpose Internet Mail Extensions, многоцелевое расширение интернет-почты) данных, предоставляемых по данному URI.

Унифицированные идентификаторы ресурсов, используемые в поставщике содержимого, напоминают не только веб-сайты, REST и сетевые службы, но и обладают сходством с именами процедур, хранимых в базах данных. Хранимые процедуры предоставляют доступ серверного типа к исходным реляционным данным. Сходство URI и хранимых процедур заключается в том, что вызовы URI, направляемые к поставщику содержимого, возвращают курсор. Однако между поставщиками содержимого и хранимыми процедурами есть различие. Оно заключается в том, что информация, играющая роль ввода при вызове службы, обычно интегрирована в URI поставщика содержимого.

Мы провели эти параллели, чтобы показать, что область функций поставщиков содержимого достаточно широка.

Структура унифицированных идентификаторов содержимого (Content URI) в Android

Мы сравнили поставщик содержимого с веб-сайтом, так как поставщик содержимого отвечает на входящие URI. Это значит, что для получения данных из поставщика содержимого нужно просто активировать URI. Однако при работе с поставщиком содержимого найденные таким образом данные представлены как набор строк и столбцов и образуют объект Android cursor. В данном контексте мы рассмотрим структуру URI, которую можно использовать для получения данных.

Унифицированные идентификаторы содержимого (Content URI) в Android напоминают HTTP URI, но начинаются с content и строятся по следующему образцу:

```
content://*/*/*
```

или

```
content://authority-name/path-segment1/path-segment2/etc…
```

Вот пример URI, при помощи которого в базе данных идентифицируется запись, имеющая номер 23:

```
content://com.google.provider.NotePad/notes/23
```

После content: в URI содержится унифицированный идентификатор источника, который используется для нахождения поставщика содержимого в соответ-

ствующем реестре. В предыдущем примере часть URI `com.google.provider.NotePad` представляет собой источник.

`/notes/23` — это раздел пути (path section), специфичный для каждого отдельного поставщика содержимого. Фрагменты `notes` и `23` раздела пути называются сегментами пути (path segments). Одной из функций поставщика содержимого является документирование и интерпретация раздела и сегментов пути, содержащихся в URI.

Обычно разработчик поставщика содержимого внедряет эти функции путем объявления констант в классе или интерфейсе Java в пакете реализации данного поставщика содержимого. В дальнейшем первый раздел пути может указывать на коллекцию объектов. Например, `/notes` указывает на коллекцию записей (или каталог с записями), а `/23` — на определенную запись.

Если URI имеет такой вид, это означает, что поставщик содержимого должен находить строки, определяемые данным URI. Кроме того, поставщик содержимого должен модифицировать содержимое URI, применяя один из методов изменения состояния: вставку (insert), обновление (update) и удаление (delete).

Структурирование MIME-типов в Android

Как веб-сайт возвращает тип MIME для заданной гиперссылки (это позволяет браузеру активировать программу, предназначенную для просмотра того или иного типа контента), так и в поставщике содержимого предусмотрена возможность возвращения типа MIME для заданного URI. Благодаря этому достигается определенная гибкость при просмотре данных. Если мы знаем, данные какого именно типа получим, то можем выбрать одну или несколько программ, предназначенных для представления таких данных. Например, если на жестком диске компьютера есть текстовый файл, мы можем выбрать несколько редакторов, которые способны его отобразить. В некоторых операционных системах даже можно предоставить возможность выбора редактора из списка.

Типы MIME работают в Android почти так же, как и в HTTP. Вы запрашиваете у поставщика содержимого тип MIME определенного поддерживаемого им URI, и поставщик содержимого возвращает двухчастную последовательность символов, идентифицирующую тип MIME в соответствии с конвенциями, принятыми в веб. Стандарт типов MIME описан на следующем сайте: **http://tools.ietf.org/html/rfc2046**.

В соответствии со спецификацией типа MIME обозначение MIME состоит из двух частей: типа и подтипа. Ниже приведены примеры некоторых известных пар типов и подтипов MIME:

```
text/html
text/css
text/xml
text/vnd.curl
application/pdf
application/rtf
application/vnd.ms-excel
```

Полный список зарегистрированных типов и подтипов доступен на сайте Полномочного органа по цифровым адресам в Интернете (Internet Assigned Numbers Authority, IANA): **http://www.iana.org/assignments/media-types**.

Основные зарегистрированные типы содержимого:

```
application
audio
example
image
message
model
multipart
text
video
```

Каждому из основных типов соответствуют подтипы. Но если производитель использует патентованные форматы данных, то название подтипа начинается с vnd. Например, таблицы Microsoft Excel идентифицируются подтипом vnd.ms-excel, а PDF считается форматом, не связанным с конкретным поставщиком, и представляется как есть, без специального префикса, обозначающего производителя.

Названия некоторых подтипов начинаются с x-; это нестандартные подтипы, которые можно не регистрировать. Они считаются частными ресурсами, которые в двухстороннем порядке оговариваются между контрагентами. Вот несколько примеров:

```
application/x-tar
audio/x-aiff
video/x-msvideo
```

В Android применяется схожий принцип для определения типов MIME. Обозначение vnd в типах MIME в Android означает, что данные типы и подтипы являются нестандартными, зависящими от производителя. Для обеспечения уникальности в Android типы и подтипы разграничиваются при помощи нескольких компонентов, как и доменные имена. Кроме того, типы MIME в Android, соответствующие каждому типу содержимого, существуют в двух формах: для одиночной записи и для нескольких записей.

Для одиночной записи тип MIME выглядит следующим образом:

```
vnd.android.cursor.item/vnd.yourcompanyname.contenttype
```

При использовании с набором строк тип MIME выглядит так:

```
vnd.android.cursor.dir/vnd.yourcompanyname.contenttype
```

Приведем пару примеров:

```
// одиночная запись
vnd.android.cursor.item/vnd.google.note

// коллекция или каталог с записями
vnd.android.cursor.dir/vnd.google.note
```

ПРИМЕЧАНИЕ

Здесь подразумевается, что в Android встроена способность различения «каталога» элементов и отдельного элемента. Но вы как программист можете экспериментировать только с подтипами. Например, такие сущности, как элементы управления списком, действуют на основании информации, возвращаемой курсором, это означает — на основании одного из «основных» типов MIME.

Типы MIME широко используются в Android, в частности при работе с намерениями, когда система определяет по MIME-типу данных, какое именно явление следует активировать. Типы MIME всегда воспроизводятся поставщиками содержимого на основании соответствующих URI. Работая с типами MIME, необходимо не упускать из виду три аспекта.

- Тип и подтип должны быть уникальными для того типа содержимого, который они представляют. Выбор типа практически полностью зависит от вас, как уже было указано. Обычно это каталог с элементами или отдельный элемент. В контексте Android разница между каталогом и элементом может быть не такой очевидной, как кажется на первый взгляд.
- Если тип или подтип не являются стандартными, им должен предшествовать префикс vnd (обычно это касается конкретных видов записи).
- Обычно типы и подтипы относятся к определенному пространству имен в соответствии с вашими нуждами.

Необходимо еще раз подчеркнуть этот момент: основной тип MIME для коллекции элементов, возвращаемый командой cursor в Android, всегда должен иметь вид vnd.android.cursor.dir, а основной тип MIME для одиночного элемента, находимый командой cursor в Android, — вид vnd.android.cursor.item. Если речь идет о подтипе, то поле для маневра расширяется, как в случае с vnd.google.note; после компонента vnd. вы можете свободно выбирать любой устраивающий вас подтип.

Считывание данных при помощи URI

Как вы теперь знаете, для получения данных от поставщика содержимого нужно использовать URI, предоставляемый этим поставщиком содержимого. Поскольку URI, определяемые конкретным поставщиком содержимого, являются уникальными для данного поставщика содержимого, эти URI необходимо документировать и предоставлять программистам, чтобы они знали и могли вызывать эти URI. Поставщики содержимого, используемые с Android, решают эту задачу путем определения констант, представляющих строки URI.

Рассмотрим три следующих URI, определенных вспомогательными классами, которые входят в состав инструментария Android SDK:

```
MediaStore.Images.Media.INTERNAL_CONTENT_URI
MediaStore.Images.Media.EXTERNAL_CONTENT_URI
Contacts.People.CONTENT_URI
```

Эквивалентные текстовые строки URI будут выглядеть следующим образом:

```
content://media/internal/images
content://media/external/images
content://contacts/people/
```

Поставщик содержимого MediaStore определяет два URI, а поставщик содержимого Contacts — один URI. Как видите, эти константы определяются по иерархической схеме. Например, URI содержимого для контактов обозначается как Contacts.People.CONTENT_URI. Это объясняется тем, что база данных с контактами может содержать много таблиц, в которых будут представлены отдельные контакты (Contact). People — это одна из таблиц или коллекция. Каждый первичный объект (primary entity) базы данных может иметь собственный URI содержимого, но этот URI все равно должен иметь тот же корень, что и базовое имя источника (например, contacts://contacts в случае с поставщиком содержимого contacts).

ПРИМЕЧАНИЕ

В ссылке Contacts.People.CONTENT_URI, Contacts — это пакет java, а People — интерфейс, находящийся внутри этого пакета.

При использовании таких URI код для получения отдельно взятой строки из people поставщика содержимого contacts будет выглядеть так:

```
Uri peopleBaseUri = Contacts.People.CONTENT_URI;
Uri myPersonUri = peopleBaseUri.withAppendedId(Contacts.People.CONTENT_URI, 23);

// Запрос данной записи.
// managedQuery - это метод, относящийся к классу Activity.
Cursor cur = managedQuery(myPersonUri, null, null, null);
```

Обратите внимание, как Contacts.People.CONTENT_URI предварительно задается в качестве константы в классе People. В данном примере код извлекает корневой URI, добавляет к нему конкретный персональный ID и вызывает метод managedQuery.

В рамках запроса такого URI можно задать порядок сортировки, столбцы для выделения и предложение Where. Эти дополнительные параметры в приведенном выше примере имеют значение null.

ПРИМЕЧАНИЕ

В поставщике содержимого должны быть перечислены поддерживаемые столбцы. Это делается при помощи применения набора интерфейсов или перечисления названий столбцов как констант. Однако класс или интерфейс, в котором задаются константы для столбцов, должен содержать информативные названия столбцов, из которых становится понятен тип каждого столбца. Это делается путем соблюдения соглашения о наименованиях (naming convention), добавления комментариев или составления документации, так как формального метода для определения типов столбцов при помощи констант не выработано.

В листинге 3.20 показано, как получить курсор с определенным списком столбцов из таблицы People поставщика содержимого contacts на базе предыдущего примера.

Листинг 3.20. Получение курсора от поставщика содержимого

```
// Массив, в котором перечисляются столбцы, которые необходимо вернуть.
string[] projection = new string[] {
   People._ID,
   People.NAME,
```

```
    People.NUMBER,
};

// Получение базового URI для таблицы People
// поставщика содержимого Contacts.
// Например, content://contacts/people/
Uri mContactsUri = Contacts.People.CONTENT_URI;

// Наилучший способ вернуть запрос; возвращает управляемый запрос.
Cursor managedCursor = managedQuery( mContactsUri,
                projection, // какие колонки вернуть
                null, // предложение WHERE
                Contacts.People.NAME + " ASC"); // предложение Order-by
```

Обратите внимание: projection — это просто массив строк, в котором перечислены названия столбцов. Таким образом, не зная, какая именно информация содержится в каких столбцах, вам будет сложно создать projection. Нужно найти названия столбцов в том же классе, из которого получен URI. В данном случае это класс People. Рассмотрим другие названия столбцов, определенные в этом классе:

```
CUSTOM_RINGTONE
DISPLAY_NAME
LAST_TIME_CONTACTED
NAME
NOTES
PHOTO_VERSION
SEND_TO_VOICE_MAIL
STARRED
TIMES_CONTACTED
```

Более подробную информацию обо всех этих столбцах можно узнать из документации по SDK, где рассматривается класс android.provider.Contacts.PeopleColumns, доступный по следующей ссылке: **http://code.google.com/android/reference/android/provider/Contacts.PeopleColumns.html**.

Как уже говорилось выше, в базе данных, например в contacts, содержится несколько таблиц, для каждой из которых предусмотрен класс или интерфейс, описывающий столбцы таблицы и их типы. Давайте рассмотрим пакет android.providers.Contacts, документация по которому содержится по следующей ссылке: **http://code.google.com/android/reference/android/provider/Contacts.html**.

В этом пакете находятся следующие вложенные классы или интерфейсы:

```
ContactMethods
Extensions
Groups
Organizations
People
Phones
Photos
Settings
```

Каждый класс назван так же, как и определенная таблица в базе данных contacts.db, и каждая таблица отвечает за описание собственной структуры URI. Кроме того,

для каждого класса определяется соответствующий интерфейс Columns, идентифицирующий названия столбцов, например PeopleColumns.

Снова рассмотрим возвращенный cursor: в нем содержится ноль или более записей. Названия, порядок и тип столбцов зависят от поставщика содержимого. Однако каждая возвращаемая строка имеет заданный по умолчанию столбец, называемый _id, в котором представлен ID этой строки.

Работа с курсором

Перед тем как получить доступ к курсору Android, необходимо изучить некоторую информацию о нем.

- Курсор — это набор строк.
- Для доступа курсора вы должны использовать moveToFirst(), так как курсор размещается перед первой строкой.
- Вы должны знать названия столбцов.
- Вы должны знать типы столбцов.
- Все методы доступа к массивам основываются на номере столбца, поэтому сначала нужно преобразовать название столбца в номер столбца.
- Курсор является случайным (random cursor) — вы можете переходить вперед, назад и со строки на строку.
- Поскольку курсор является случайным, у него можно запрашивать количество строк (row count).

В курсоре Android существуют методы, позволяющие переходить от одного компонента курсора к другому. В листинге 3.21 показано, как проверить, пуст ли курсор, и как просмотреть курсор строка за строкой, если в нем есть информация.

Листинг 3.21. Навигация по курсору при помощи цикла while

```
if (cur.moveToFirst() == false)
{
   // строк нет, курсор пуст
   return;
}

// Курсор уже указывает на первую строку.
// Давайте получим доступ к нескольким столбцам.
int nameColumnIndex = cur.getColumnIndex(People.NAME);
String name = cur.getString(nameColumnIndex);

// Давайте узнаем, как просмотреть курсор при помощи цикла.

while(cur.moveToNext())
{
   // курсор перемещен успешно
   // поля доступа
}
```

В начале листинга 3.21 предполагалось, что курсор расположен перед первой строкой. Чтобы разместить курсор перед первой строкой, мы использовали метод moveToFirst() для объекта cursor. Если курсор пуст, этот метод возвращает false. Затем мы применяем метод moveToNext() несколько раз для перемещения по курсору.

Чтобы помочь вам находить курсор, Android предоставляет три следующих метода:

```
isBeforeFirst()
isAfterLast()
isClosed()
```

Используя эти методы, вы можете также применять для движения по курсору цикл for, как в листинге 3.22, а не цикл while, как в листинге 3.21.

Листинг 3.22. Навигация по курсору с применением цикла

```
for(cur.moveToFirst();!cur.isAfterLast();cur.moveToNext())
{
 int nameColumn = cur.getColumnIndex(People.NAME);
 int phoneColumn = cur.getColumnIndex(People.NUMBER);
 String name = cur.getString(nameColumn);
 String phoneNumber = cur.getString(phoneColumn);
}
```

Работа с предложением where

В поставщиках содержимого предусмотрены две возможности передачи предложения where:

- при помощи URI;
- используя комбинацию предложения string и набор заменяемых аргументов вида «строка — массив» (string-array).

Оба этих подхода будут рассмотрены на примерах кода.

Передача предложения where через URI

Предположим, нам нужно получить запись с ID 23 из базы данных Google Notes. Воспользуйтесь кодом, приведенным в листинге 3.23, чтобы получить курсор, содержащий одну строку, соответствующую строке 23 в таблице записей.

Листинг 3.23. Передача предложений SQL WHERE при помощи URI

```
Activity someActivity;
//..инициализация someActivity
String noteUri = "content://com.google.provider.NotePad/notes/23";
Cursor managedCursor = someActivity.managedQuery( noteUri,
                    projection, // столбцы, которые следует вернуть
                    null, // предложение WHERE
                    null); // предложение Order-by
```

Мы оставили аргумент clause метода managedQuery равным нулю (null), так как в данном случае мы предполагали, что поставщик содержимого достаточно интеллектуален для того, чтобы найти id нужной нам книги. Этот id интегрирован в URI.

Таким образом, мы использовали URI как носитель для передачи предложения where. Механизм становится более понятным, если обратить внимание на то, как поставщик записей реализует соответствующий метод запроса. Рассмотрим фрагмент кода из этого метода:

```
// Получить id записи из входящего URI, имеющего вид
// content://.../notes/23.
int noteId = uri.getPathSegments().get(1);

// приказать построителю запросов создать запрос
// указать имя таблицы
queryBuilder.setTables(NOTES_TABLE_NAME);

// Использовать noteid для вставки предложения where.
queryBuilder.appendWhere(Notes._ID + "=" + );
```

Обратите внимание на то, как id записи извлекается из URI. Класс Uri, представляющий водящий аргумент uri, использует специальный метод для извлечения фрагментов URI после корня: content://com.google.provider.NotePad. Эти фрагменты называются сегментами пути (path segments). Они представляют собой последовательности символов, расположенные между разделительными символами /, например /seg1/seg3/seg4/, и индексируются по своим позициям. В приведенном здесь URI первый сегмент пути — 23. Затем мы используем этот ID, прикрепляя его к предложению where, указанному в классе QueryBuilder. В итоге имеем следующий эквивалентный оператор выбора (select statement):

```
select * from notes where _id = 23
```

ПРИМЕЧАНИЕ

Классы Uri и UriMatcher используются для того, чтобы идентифицировать URI и извлекать из них параметры (UriMatcher будет рассмотрен далее, в подразделе «Использование UriMatcher для определения URI»). SQLiteQueryBuilder — это вспомогательный класс из android.database.sqlite, позволяющий создавать SQL-запросы, которые позже будут выполняться SQLiteDatabase в экземпляре базы данных SQLite.

Использование явных предложений WHERE

Теперь, когда вы знаете, как использовать URI для отправки предложения where, рассмотрим и другой метод, применяемый в Android, чтобы посылать список явных столбцов (explicit columns) и соответствующих им значений в виде предложения where. Чтобы исследовать этот метод, еще раз обратимся к методу managedQuery класса Activity, с которым мы работали в листинге 3.23. Вот его схема:

```
public final Cursor managedQuery(Uri uri,
   String[] projection,
   String selection,
   String[] selectionArgs,
   String sortOrder)
```

Обратите внимание на аргумент selection типа string. Данная строка выбора представляет собой фильтр (то есть предложение where), в котором объявляется,

какие строки должны возвращаться в таком формате, как предложение SQL WHERE (исключая само предложение WHERE). При передаче значения null будут возвращены все строки, соответствующие указанному URI. В строке выбора могут находиться символы ?, которые будут заменяться значениями selectionArgs в том порядке, в котором они появляются в выборке. Значения будут связаны как String.

Поскольку существует два способа задать предложение where, может быть сложно определить, как именно поставщик содержимого задействует определенные предложения where, и какие предложения where будут иметь приоритет при одновременном применении обоих методов.

Например, можно сделать запрос записи, имеющей ID 23, воспользовавшись поочередно обоими методами:

```
// метод с URI
managedQuery("content://com.google.provider.NotePad/notes/23"
,null
,null
,null
,null);
```

или

```
// явное предложение where
managedQuery("content://com.google.provider.NotePad/notes"
,null
,"_id=?"
,new String[] {23}
,null);
```

При работе с предложениями where принято использовать метод с применением URI как стандартный, а метод, в котором задействуются явные предложения where, оставлять для особых случаев.

Вставка записей

Итак, мы изучили, как получать данные от поставщиков содержимого, пользуясь URI. Теперь обратимся к операциям вставки (insert), обновления (update) и удаления (delete). Начнем с insert.

В Android используется класс android.content.ContentValues, содержащий значения для отдельной записи, которую предстоит вставить. ContentValues — это словарь, состоящий из пар «ключ/значение», которые во многом похожи на названия столбцов и значения этих названий. Для вставки записи сначала нужно занести ее в ContentValues, а потом приказать android.content.ContentResolver вставить эту запись, используя URI.

ПРИМЕЧАНИЕ

Вам потребуется найти ContentResolver, так как на этом уровне абстракции вы не приказываете базе данных вставить запись; вы говорите вставить запись в поставщик содержимого, идентифицируемый по URI. ContentResolver getCount() отвечает за разрешение ссылки URI в нужный поставщик содержимого и за последующую передачу объекта ContentValues данному конкретному поставщику содержимого.

Ниже приведен пример заполнения отдельно взятой строки записей, находящейся в ContentValues при подготовке этой строки к вставке:

```
ContentValues values = new ContentValues();
values.put("title", "New note");
values.put("note","This is a new note");

// Теперь объект values может быть вставлен.
```

Хотя мы жестко закодировали названия столбцов таблицы, вы можете использовать вместо них константы, заданные в приложении NotePad. Получить ссылку на ContentResolver можно, запросив класс Activity:

```
ContentResolver contentResolver = activity.getContentResolver();
```

Теперь все, что нам нужно, — это URI, в котором мы сообщим ContentResolver о необходимости вставить строку. Такие URI определяются в отдельном классе в соответствии с таблицей Notes. В примере с NotePad этот URI имеет следующий вид:

```
Notepad.Notes.CONTENT_URI
```

Мы можем взять этот URI и имеющиеся у нас ContentValues и сделать вызов, при помощи которого мы вставим строку:

```
Uri uri = contentResolver.insert(Notepad.Notes.CONTENT_URI, values);
```

Данный вызов возвращает URI, указывающий на только что созданную запись. Этот возвращенный URI будет иметь следующую структуру:

```
Notepad.Notes.CONTENT_URI/new_id
```

Добавление файла в поставщик содержимого

Иногда может возникнуть необходимость сохранить файл в базе данных. Обычно для этого файл сохраняется на диске, а затем в базе данных обновляется запись, указывающая на файл с соответствующим названием.

Android автоматизирует работу такого протокола, задавая специальную процедуру для сохранения и нахождения таких файлов. В Android используется соглашение, по которому ссылка на имя файла сохраняется в записи с зарезервированным именем столбца _data.

Когда запись вставляется в такую таблицу, Android возвращает URI вызывающему оператору. Как только вы сохраните запись таким методом, вам будет нужно сопроводить эту запись файлом, который будет сохранен вместе с ней. Для реализации этой функции Android позволяет ContentResolver взять URI записи, содержащейся в базе данных, и возвратить выходной поток с возможностью записи (writable output stream). «За кулисами» Android выделяет специальный внутренний файл и сохраняет ссылку на это имя файла в массиве _data.

Если бы мы попытались расширить пример с NotePad, и добавить функцию сохранения изображения с заданной записью, мы могли бы создать дополнительный столбец под названием _data и осуществить операцию вставки в первый раз, чтобы

возвратить URI. В следующем фрагменте кода демонстрируется данная часть протокола:

```
ContentValues values = new ContentValues();
values.put("title", "New note");
values.put("note","This is a new note");

// Использовать content resolver для вставки записи.
ContentResolver contentResolver = activity.getContentResolver();
Uri newUri = contentResolver.insert(Notepad.Notes.CONTENT_URI, values);
```

Как только в вашем распоряжении окажется URI этой записи, следующим кодом можно будет приказать ContentResolver получить ссылку на выходной поток файла:

```
....
// Использовать content resolver, чтобы напрямую получить выходной поток.
// ContentResolver скрывает доступ к массиву _data, где
// он сохраняет реальную ссылку на файл.
OutputStream outStream = activity.getContentResolver().openOutputStream(newUri);
someSourceBitmap.compress(Bitmap.CompressFormat.JPEG, 50, outStream);
outStream.close();
```

Затем код может записать информацию в этот выходной поток.

Обновления и удаления

Мы уже обсудили, как работать с запросами и операциями вставки. Операции обновления и удаления также очень просты. Операция обновления похожа на операцию вставки, при этом измененные значения столбцов передаются посредством объекта ContentValues. Ниже приведена схема метода обновления, используемого объектом ContentResolver:

```
int numberOfRowsUpdated =
activity.getContentResolver().update(
   Uri uri,
   ContentValues values,
   String whereClause,
   String[] selectionArgs )
```

Аргумент whereClause распространяет обновление на соответствующие строки. Схема метода delete выглядит схожим образом:

```
int numberOfRowsDeleted =
activity.getContentResolver().delete(
   Uri uri,
   String whereClause,
   String[] selectionArgs )
```

Разумеется, при применении метода delete вам не потребуется аргумент ContentValues, так как не нужно будет указывать столбцы при удалении записи.

Почти все вызовы, исходящие от managedQuery и ContentResolver, в конечном счете направляются к классу provider. Зная, как поставщик содержимого реализует

каждый из этих методов, мы вполне можем понять, как эти методы будут использоваться клиентом. В следующем разделе мы рассмотрим пример разработки поставщика содержимого, называемого BookProvider.

Реализация поставщиков содержимого

Вы изучили, как обмениваться информацией с поставщиком содержимого при работе с данными, но еще не знаете, как написать поставщик содержимого. Итак, чтобы это сделать, нужно дополнить android.content.ContentProvider, реализовав следующие основные методы:

```
query
insert
update
delete
getType
```

Правда, чтобы эти методы работали, нам потребуется выполнить предварительные действия. Мы подробно рассмотрим все детали реализации поставщика содержимого, описав, какие шаги вам потребуется выполнить при этом.

1. Спланировать строение базы данных, URI, названия столбцов и т. д и создать класс метаданных, в котором определим константы для всех элементов метаданных.
2. Расширить абстрактный класс ContentProvider.
3. Реализовать следующие методы: query, insert, update, delete и getType.
4. Зарегистрировать поставщик содержимого в файле описания.

Планирование базы данных

Создадим базу данных, в которой будет содержаться информация о коллекции книг, чтобы на практике изучить данный вопрос. В книжной базе данных будет находиться только одна таблица, называемая books, столбцы которой будут называться name, isbn и author. Релевантные метаданные такого рода будут определяться в специальном классе Java. Класс Java BookProviderMetaData, в котором заключены метаданные, показан в листинге 3.24. Наиболее важные элементы этого класса метаданных выделены полужирым.

Листинг 3.24. Определение метаданных для базы данных: класс BookProviderMetaData

```
public class BookProviderMetaData
{
    public static final String AUTHORITY =
            "com.androidbook.provider.BookProvider";

    public static final String DATABASE_NAME = "book.db";
    public static final int DATABASE_VERSION = 1;
    public static final String BOOKS_TABLE_NAME = "books";

    private BookProviderMetaData() {}

    // внутренний класс, описывающий BookTable
```

```
    public static final class BookTableMetaData implements BaseColumns
    {

        private BookTableMetaData() {}
        public static final String TABLE_NAME = "books";

        // определение uri и типа MIME
        public static final Uri CONTENT_URI =
                    Uri.parse("content://" + AUTHORITY + "/books");

        public static final String CONTENT_TYPE =
                    "vnd.android.cursor.dir/vnd.androidbook.book";

        public static final String CONTENT_ITEM_TYPE =
                    "vnd.android.cursor.item/vnd.androidbook.book";

        public static final String DEFAULT_SORT_ORDER = "modified DESC";

        // здесь начинаются дополнительные столбцы
        // тип строка
        public static final String BOOK_NAME = "name";

        // тип строка
        public static final String BOOK_ISBN = "isbn";

        // тип строка
        public static final String BOOK_AUTHOR = "author";

        // натуральное число из System.currentTimeMillis()
        public static final String CREATED_DATE = "created";

        // натуральное число из System.currentTimeMillis()
        public static final String MODIFIED_DATE = "modified";
    }
}
```

Данный класс BookProviderMetaData начинается с определения его источника — com.androidbook.provider.BookProvider. Воспользуемся этой строкой, чтобы зарегистрировать поставщик в файле описания Android. Данная строка образует первую часть URI, предназначенного для этого поставщика.

Затем класс продолжает работу, определяя собственную таблицу (books) как внутренний класс BookTableMetaData. Затем класс BookTableMetaData задает URI для идентификации коллекции книг. С учетом источника, указанного в предыдущем абзаце, URI для коллекции книг будет выглядеть так:

```
content://com.androidbook.provider.BookProvider/books
```

Данный URI обозначается константой:

```
BookProviderMetaData.BookTableMetaData.CONTENT_URI
```

Класс BookTableMetaData продолжает работу и задает типы MIME для коллекции книг и отдельно взятой книги. Внедренный в систему поставщик содержимого будет использовать эти константы для возвращения типов MIME входящих URI.

Затем BookTableMetaData задает набор столбцов: name, isbn, author, created (дата создания) и modified (дата внесения последних изменений).

ПРИМЕЧАНИЕ

Типы данных, представляемые в конкретных столбцах вашей базы данных, можно обозначить непосредственно в коде при помощи комментариев.

Класс метаданных BookTableMetaData также наследует свойства класса BaseColumns, в котором предоставляется стандартное поле _id, содержащее ID строки. Располагая такими определениями метаданных, мы можем перейти к внедрению поставщика содержимого в систему.

Расширение ContentProvider

Для реализации выбранного нами в качестве примера поставщика содержимого BookProvider потребуется дополнить класс ContentProvider и заменить метод onCreate(), чтобы создать базу данных, а затем реализовать методы query, insert, update, delete и getType. В этом подразделе рассмотрим, как создать и настроить базу данных, а в следующих разделах отдельно изучим каждый метод: query, insert, update, delete и getType.

Методу query требуется набор столбцов, который он должен вернуть. Это напоминает предложение select, которому нужны названия столбцов вместе с соответствующими им as (иногда они называются синонимами). В Android используется объект map, вызывающий карту projection, в которой представляются названия этих столбцов и их синонимы. Нам потребуется настроить эту карту, чтобы позже мы могли использовать ее при реализации метода query. В коде, предназначенном для внедрения в систему поставщика содержимого (листинг 3.25), показано, как это сделать.

Большинство реализуемых нами методов будут использовать URI в качестве исходных данных. Хотя начало всех URI, на которые может ответить такой поставщик содержимого, должно строиться в соответствии со стандартным шаблоном, оканчиваться URI будут по-разному — как и адреса веб-сайтов. Каждый URI, пусть он и начинается с шаблона, должен быть уникальным, чтобы по нему можно было идентифицировать различные данные и документы. Рассмотрим этот момент на примере:

```
Uri1: content://com.androidbook.provider.BookProvider/books
Uri2: content://com.androidbook.provider.BookProvider/books/12
```

Таким образом, Book Provider должен отличать друг от друга разные URI. Это простой случай. Если бы в таком поставщике книг находились не только книги, но и другие объекты, потребовалось бы больше URI, чтобы идентифицировать весь спектр объектов.

В поставщике содержимого должен быть механизм, при помощи которого можно было бы отличать один URI от другого. В Android эту задачу выполняет класс UriMatcher. Итак, нам нужно настроить объект и все варианты его URI. Этот момент

будет показан в коде листинга 3.25 после того фрагмента, в котором создается проекционная карта. Мы более подробно рассмотрим класс UriMatcher в подразделе «Использование UriMatcher для определения URI», но на данный момент остановимся на том, что код, показанный в этом листинге, позволяет поставщику содержимого отличать один URI от другого.

Наконец, код из листинга 3.25 заменяет метод onCreate(), чтобы упростить создание базы данных. Мы разделили код специально выделенными комментариями, чтобы обозначить три области, о которых пойдет речь:

- настройка проекции столбцов;
- настройка UriMatcher;
- создание базы данных.

Листинг 3.25. Реализация поставщика содержимого BookProvider

```
public class BookProvider extends ContentProvider
{
   // Создание проекционной карты для столбцов.
   // Проекционные карты похожи на сущности "as" в sql
   // оператор, при помощи которого можно переименовывать
   // столбцы.
   private static HashMap<String, String> sBooksProjectionMap;
   static
   {
      sBooksProjectionMap = new HashMap<String, String>();
      sBooksProjectionMap.put(BookTableMetaData._ID,
            BookTableMetaData._ID);

      // название, ISBN, автор
      sBooksProjectionMap.put(BookTableMetaData.BOOK_NAME
            , BookTableMetaData.BOOK_NAME);
      sBooksProjectionMap.put(BookTableMetaData.BOOK_ISBN
            , BookTableMetaData.BOOK_ISBN);
      sBooksProjectionMap.put(BookTableMetaData.BOOK_AUTHOR
            , BookTableMetaData.BOOK_AUTHOR);

      //дата создания, дата внесения изменений
      sBooksProjectionMap.put(BookTableMetaData.CREATED_DATE
            , BookTableMetaData.CREATED_DATE);
      sBooksProjectionMap.put(BookTableMetaData.MODIFIED_DATE
            , BookTableMetaData.MODIFIED_DATE);
   }

// Обеспечение механизма для идентификации всех входящих шаблонов uri.
private static final UriMatcher sUriMatcher;
private static final int INCOMING_BOOK_COLLECTION_URI_INDICATOR = 1;
private static final int INCOMING_SINGLE_BOOK_URI_INDICATOR = 2;
static {
     sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
     sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
           , "books"
```

```
            , INCOMING_BOOK_COLLECTION_URI_INDICATOR);

      sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
            , "books/#",
            INCOMING_SINGLE_BOOK_URI_INDICATOR);
      }

// Работа с обратным вызовом OnCreate.

      private DatabaseHelper mOpenHelper;

      @Override
      public boolean onCreate() {
         mOpenHelper = new DatabaseHelper(getContext());
         return true;
      }

      private static class DatabaseHelper extends SQLiteOpenHelper {

         DatabaseHelper(Context context) {
            super(context, BookProviderMetaData.DATABASE_NAME, null
                  , BookProviderMetaData.DATABASE_VERSION);
         }

// Создание базы данных.
         @Override
         public void onCreate(SQLiteDatabase db) {
            db.execSQL("CREATE TABLE "
               + BookTableMetaData.TABLE_NAME + " ("
               + BookProviderMetaData.BookTableMetaData._ID
               + " INTEGER PRIMARY KEY,"
               + BookTableMetaData.BOOK_NAME + " TEXT,"
               + BookTableMetaData.BOOK_ISBN + " TEXT,"
               + BookTableMetaData.BOOK_AUTHOR + " TEXT,"
               + BookTableMetaData.CREATED_DATE + " INTEGER,"
               + BookTableMetaData.MODIFIED_DATE + " INTEGER"
               + ");");
         }

// Работа с изменениями версий.
         @Override
         public void onUpgrade(SQLiteDatabase db, int oldVersion,
         int newVersion) {
            Log.w(TAG, "Upgrading database from version "
               + oldVersion + " to "
               + newVersion + ", which will destroy all old data");
            db.execSQL("DROP TABLE IF EXISTS "
               + BookTableMetaData.TABLE_NAME);
            onCreate(db);
         }
   }
```

Выполнение соглашений, связанных с типами MIME

Поставщик содержимого BookProvider должен включать метод getType(), предназначенный для возвращения типа MIME каждого отдельного URI. Этот метод, как и многие другие методы, связанные с поставщиками содержимого, переопределяется в зависимости от входящего URI. В результате первой задачей метода getType() является определение типа URI. Соответствует ли данный уникальный идентификатор ресурса коллекции книг или отдельной книге?

Как было указано в предыдущем подразделе, мы используем UriMatcher, чтобы определить тип URI. В зависимости от типа URI, в классе BookTableMetaData определяются константы для типов MIME, предназначенные для возврата каждого конкретного URI. Без лишних разговоров мы представим весь код, при помощи которого реализуется метод getType() (листинг 3.26).

Листинг 3.26. Реализация метода getType()

```
@Override
public String getType(Uri uri) {
   switch (sUriMatcher.match(uri)) {
   case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
      return BookTableMetaData.CONTENT_TYPE;

   case INCOMING_SINGLE_BOOK_URI_INDICATOR:
      return BookTableMetaData.CONTENT_ITEM_TYPE;

   default:
      throw new IllegalArgumentException("Unknown URI " + uri);
   }
}
```

Реализация метода query

Метод query отвечает в поставщике содержимого за возврат набора строк в зависимости от входящего URI и предложения where.

Как и другие методы, query использует UriMatcher для идентификации типа URI. Если URI соответствует одиночному элементу, метод извлекает ID книги из входящего URI следующим образом.

1. Извлекает сегменты пути при помощи getPathSegments().
2. Просматривает URI для получения первого сегмента пути, который является ID книги.

Затем метод query использует projections, созданные нами в листинге 3.25, для идентификации возвращенных столбцов. Закончив работу, query возвращает курсор вызывающему оператору. При выполнении этого процесса метод query использует объект SQLiteQueryBuilder для формулирования и выполнения запроса (листинг 3.27).

Листинг 3.27. Реализация метода query()

```
@Override
public Cursor query(Uri uri, String[] projection, String selection
                    , String[] selectionArgs, String sortOrder)
```

```
{
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

    switch (sUriMatcher.match(uri))
    {
        case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
        qb.setTables(BookTableMetaData.TABLE_NAME);
        qb.setProjectionMap(sBooksProjectionMap);
        break;

        case INCOMING_SINGLE_BOOK_URI_INDICATOR:
        qb.setTables(BookTableMetaData.TABLE_NAME);
        qb.setProjectionMap(sBooksProjectionMap);
        qb.appendWhere(BookTableMetaData._ID + "="
                + uri.getPathSegments().get(1));
        break;

        default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }

    // Если порядок сортировки не задан по умолчанию.
    String orderBy;
    if (TextUtils.isEmpty(sortOrder)) {
       orderBy = BookTableMetaData.DEFAULT_SORT_ORDER;
    } else {
       orderBy = sortOrder;
    }

    // Выход к базе данных и выполнение запроса.
    SQLiteDatabase db =
        mOpenHelper.getReadableDatabase();
    Cursor c = qb.query(db, projection, selection,
                selectionArgs, null, null, orderBy);
    int i = c.getCount();

    // Сообщение курсору, за каким uri наблюдать,
    // чтобы было известно обо всех изменениях исходных данных.
    c.setNotificationUri(getContext().getContentResolver(), uri);
    return c;
  }
```

Реализация метода insert

Используемый поставщиком содержимого метод insert отвечает за вставку записи в основную базу данных и за последующее возвращение URI, указывающего на новую запись.

Как и другие методы, insert использует UriMatcher для идентификации типа URI. Сначала код проверяет, соответствует ли URI типу, используемому для ука-

зания на коллекцию объектов. Если это не так, код генерирует исключительную ситуацию (листинг 3.28).

Затем код проверяет опциональные и обязательные параметры столбцов. Если в некоторых столбцах пропущены значения, код может вставить значения, заданные по умолчанию.

После этого код использует объект SQLiteDatabase для вставки новой записи и возвращает ее ID. Наконец, код составляет новый URI, в котором применяет ID, возвращенный из базы данных.

Листинг 3.28. Реализация метода insert()

```
@Override
public Uri insert(Uri uri, ContentValues values) {
   // проверка запрошенного uri
   if (sUriMatcher.match(uri) != INCOMING_BOOK_COLLECTION_URI_INDICATOR) {
      throw new IllegalArgumentException("Unknown URI " + uri);
   }

   Long now = Long.valueOf(System.currentTimeMillis());

   // проверка полей, предназначенных для ввода
   // Убеждаемся, что все поля заполнены.
   if (values.containsKey(BookTableMetaData.CREATED_DATE) == false) {
      values.put(BookTableMetaData.CREATED_DATE, now);
   }

   if (values.containsKey(BookTableMetaData.MODIFIED_DATE) == false) {
      values.put(BookTableMetaData.MODIFIED_DATE, now);
   }

   if (values.containsKey(BookTableMetaData.BOOK_NAME) == false) {
      throw new SQLException(
         "Failed to insert row because Book Name is needed " + uri);
   }

   if (values.containsKey(BookTableMetaData.BOOK_ISBN) == false) {
      values.put(BookTableMetaData.BOOK_ISBN, "Unknown ISBN");
   }

   if (values.containsKey(BookTableMetaData.BOOK_AUTHOR) == false) {
      values.put(BookTableMetaData.BOOK_ISBN, "Unknown Author");
   }

   SQLiteDatabase db = mOpenHelper.getWritableDatabase();
   long rowId = db.insert(BookTableMetaData.TABLE_NAME
                   , BookTableMetaData.BOOK_NAME, values);
   if (rowId > 0) {
      Uri insertedBookUri = ContentUris.withAppendedId(
                      BookTableMetaData.CONTENT_URI, rowId);
```

```
        getContext().getContentResolver().notifyChange(
                    insertedBookUri, null);
        return insertedBookUri;
    }

    throw new SQLException("Failed to insert row into " + uri);
}
```

Реализация метода update

Метод update, который используется в поставщике содержимого, отвечает за обновление записи на основе значений, переданных в столбце, а также с учетом полученного предложения where. Затем метод update возвращает количество строк, в которые были внесены обновления.

Подобно другим методам, update использует UriMatcher для идентификации типа URI. Если URI соответствует коллекции, то предложение where пропускается, так что он может включать любое требуемое количество записей. Если URI соответствует одиночной записи, то ID книги извлекается из URI и указывается в виде дополнительного предложения where. Наконец, код возвращает количество обновленных записей (листинг 3.29). В главе 12 подробно объяснены импликации данного метода notifyChange. Обратите внимание, как метод notifyChange позволяет вывести сообщение о том, что данные, на которые указывает этот URI, были изменены. Потенциально эту задачу можно выполнить при помощи метода insert, сообщив при вставке записи, что в "..../books" были внесены изменения.

Листинг 3.29. Реализация метода update

```
@Override
public int update(Uri uri, ContentValues values, String where, String[] whereArgs)
{
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri)) {
    case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
        count = db.update(BookTableMetaData.TABLE_NAME,
                    values, where, whereArgs);
        break;

    case INCOMING_SINGLE_BOOK_URI_INDICATOR:
        String rowId = uri.getPathSegments().get(1);
        count = db.update(BookTableMetaData.TABLE_NAME
                , values
              , BookTableMetaData._ID + "=" + rowId
          + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : "")
              , whereArgs);
        break;

    default:
        throw new IllegalArgumentException("Unknown URI " + uri);
```

```
    }

    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

Реализация метода delete

Метод delete, используемый в поставщике содержимого, отвечает за удаление записей и выполняет эту задачу на основании получаемых предложений where. Затем метод delete возвращает количество удаленных строк.

Подобно другим методам, delete использует UriMatcher для идентификации типа URI. Если URI соответствует коллекции элементов, предложение where пропускается и вы можете удалить любое требуемое количество записей. Если предложение where имеет значение null, то все записи удаляются. Если URI соответствует одиночной записи, из него извлекается ID книги и этот ID указывается в дополнительном предложении where. Наконец, код возвращает количество удаленных записей (листинг 3.30).

Листинг 3.30. Реализация метода delete()

```
@Override
public int delete(Uri uri, String where, String[] whereArgs) {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    int count;
    switch (sUriMatcher.match(uri)) {
    case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
        count = db.delete(BookTableMetaData.TABLE_NAME, where, whereArgs);
        break;

    case INCOMING_SINGLE_BOOK_URI_INDICATOR:
        String rowId = uri.getPathSegments().get(1);
        count = db.delete(BookTableMetaData.TABLE_NAME
            , BookTableMetaData._ID + "=" + rowId
            + (!TextUtils.isEmpty(where) ? " AND (" + where + ')' : "")
            , whereArgs);
        break;

    default:
        throw new IllegalArgumentException("Unknown URI " + uri);
    }
    getContext().getContentResolver().notifyChange(uri, null);
    return count;
}
```

Использование UriMatcher для определения URI

Мы уже несколько раз упоминали о классе UriMatcher, сейчас рассмотрим его подробнее. Почти все методы, используемые поставщиком содержимого, переопределяются в зависимости от полученного URI. Например, при необходимости получить отдельную книгу или список книг вызывается один и тот же метод query(). Метод

должен распознавать, какой тип URI запрашивается. Вспомогательный класс UriMatcher, применяемый в Android, помогает идентифицировать типы URI.

Ниже показан принцип работы этого класса: вы сообщаете экземпляру UriMatcher, какой шаблон URI следует ожидать. Кроме того, каждому шаблону URI присваивается уникальный номер. Когда эти шаблоны будут зарегистрированы, вы можете присвоить каждому шаблону уникальный номер.

Как уже указывалось выше, поставщик содержимого BookProvider различает два шаблона URI: для коллекции книг и для одиночной книги. Код, приведенный в листинге 3.31, регистрирует оба этих шаблона при помощи UriMatcher. Коллекции книг присваивается номер 1, а отдельной книге — номер 2 (сами шаблоны URI определяются в метаданных таблицы books).

Листинг 3.31. Регистрация шаблонов URI при помощи UriMatcher

```
private static final UriMatcher sUriMatcher;
// определение id для каждого типа uri
private static final int INCOMING_BOOK_COLLECTION_URI_INDICATOR = 1;
private static final int INCOMING_SINGLE_BOOK_URI_INDICATOR = 2;

static {
   sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
   // регистрация шаблона для books
   sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
                   , "books"
                   , INCOMING_BOOK_COLLECTION_URI_INDICATOR);
   // регистрация шаблона для single book
   sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
                   , "books/#",
                   INCOMING_SINGLE_BOOK_URI_INDICATOR);
}
```

Теперь, когда регистрация состоялась, можно рассмотреть, как UriMatcher участвует в реализации метода запроса:

```
switch (sUriMatcher.match(uri)) {
   case INCOMING_BOOK_COLLECTION_URI_INDICATOR:
   ......
   case INCOMING_SINGLE_BOOK_URI_INDICATOR:
   ......
   default:
      throw new IllegalArgumentException("Unknown URI " + uri);
}
```

Обратите внимание, как метод match возвращает то же число, которое было зарегистрировано ранее. Конструктор UriMatcher берет натуральное число для использования с корневым URI. UriMatcher возвращает это число, если в URL отсутствуют и сегменты пути, и источники. Кроме того, UriMatcher возвращает NO_MATCH, если шаблоны не совпадают. UriMatcher можно построить, не используя код, совпадающий с корневым. В таком случае Android осуществляет внутрисистемную инициализацию UriMatcher со значением NO_MATCH. Таким образом, для решения задачи, представленной в листинге 3.31, можно использовать следующий код:

```
static {
 sUriMatcher = new UriMatcher();
 sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
                    , "books"
                    , INCOMING_BOOK_COLLECTION_URI_INDICATOR);

 sUriMatcher.addURI(BookProviderMetaData.AUTHORITY
                    , "books/#",
                    INCOMING_SINGLE_BOOK_URI_INDICATOR);
}
```

Использование карт соответствий

Поставщик содержимого действует в качестве посредника между абстрактным набором столбцов и реальными столбцами, расположенными в базе данных. Однако два указанных набора столбцов могут различаться. При создании запросов необходимо обеспечить соответствие между столбцами, указанными в предложении where, указываемыми клиентом, и фактическим набором столбцов, которые присутствуют в базе данных. Такая *карта соответствий* (projection map) создается при помощи класса SQLiteQueryBuilder.

В документации по Android SDK о методе public void setProjectionMap(Map columnMap), входящем в состав класса QueryBuilder, сказано следующее.

Создает карту соответствий для запроса. Данная карта соответствий сопоставляет названия столбцов, передаваемые вызывающим оператором в запросе с названиями столбцов, имеющихся в базе данных. Эта функция полезна при переименовании столбцов, а также при устранении омонимии между названиями столбцов, возникающей в ходе выполнения соединений. Например, можно представить name как people.name. Если вы пользуетесь картой соответствий, то в ней должны содержаться все названия столбцов, которые может запросить пользователь, даже если пары «ключ — значение» этих столбцов являются одинаковыми.

Рассмотрим, как поставщик содержимого BookProvider создает карту соответствий:

```
sBooksProjectionMap = new HashMap<String, String>();
sBooksProjectionMap.put(BookTableMetaData._ID, BookTableMetaData._ID);

// имя, isbn, автор
sBooksProjectionMap.put(BookTableMetaData.BOOK_NAME
                    , BookTableMetaData.BOOK_NAME);
sBooksProjectionMap.put(BookTableMetaData.BOOK_ISBN
                    , BookTableMetaData.BOOK_ISBN);
sBooksProjectionMap.put(BookTableMetaData.BOOK_AUTHOR
                    , BookTableMetaData.BOOK_AUTHOR);

// дата создания, дата изменения
sBooksProjectionMap.put(BookTableMetaData.CREATED_DATE
                    , BookTableMetaData.CREATED_DATE);
sBooksProjectionMap.put(BookTableMetaData.MODIFIED_DATE
                    , BookTableMetaData.MODIFIED_DATE);
```

Затем построитель запросов использует переменную sBooksProjectionMap так:

```
queryBuilder.setTables(NOTES_TABLE_NAME);
queryBuilder.setProjectionMap(sNotesProjectionMap);
```

Регистрация поставщика содержимого

Наконец, нужно зарегистрировать поставщик содержимого в файле Android.Manifest.xml при помощи следующей структуры тегов:

```
<provider android:name="BooksProvider"
   android:authorities=" com.androidbook.provider.BookProvider "/>
```

На этом мы завершаем обсуждение поставщиков содержимого. В этом разделе была изучена природа уникальных идентификаторов контент-ресурсов и типы MIME, а также принцип использования SQLite при создании поставщиков содержимого, отвечающих на URI. Если основные данные в вашей базе выражены по такому принципу, любое приложение платформы Android будет работать более эффективно. Подобная возможность доступа к данным и их обновления при помощи URI, независимо от границ действующих процессов, согласуются с современными сервис-ориентированными разработками с привлечением облачных вычислений, о которых мы упоминали в главе 1. В следующем разделе мы рассмотрим намерения, которые связаны с поставщиками содержимого через URI и типы MIME. Знания, приобретенные в этом разделе, очень пригодятся вам при изучении намерений.

Намерения

Под концептом «намерение» (intent) в Android понимается совокупность нескольких идей. При помощи намерений вы можете активировать различные приложения из программы, с которой работаете в настоящий момент. Кроме того, используя намерения, можно активировать внутренние или внешние программные компоненты. Намерения можно применять для вызова событий (event raising) так, чтобы другие компоненты могли реагировать на такие вызовы по принципу, напоминающему модель «публикация и подписка» (publish-and-subscribe). От того, что собой представляет информационное наполнение намерения, зависит, какое именно событие будет активироваться этим намерением. Итак, что же такое намерение?

ПРИМЕЧАНИЕ

Что такое намерение? Самый простой ответ: «Это действие, с которым связана определенная информационная нагрузка».

Предельно просто намерение можно определить как действие, говорящее Android активировать определенный процесс. Действие, активируемое Android, зависит от того, запуск какого компонента системы определен в качестве реакции на это намерение. Допустим, мы написали следующее явление:

```
public class BasicViewActivity extends Activity
{
   @Override
```

```
    public void onCreate(Bundle savedInstanceState)
    {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.some-view);
    }
} // eof-class
```

Android дает воможность зарегистрировать явление в файле описания, позволив другим приложениям активировать это явление. Регистрация выглядит следующим образом:

```
<activity android:name="BasicViewActivity"
          android:label="Basic View Tests">
<intent-filter>
   <action android:name="com.androidbook.intent.action.ShowBasicView"/>
   <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
</activity>
```

В данном случае при регистрации описывается не только явление, но и действие, которое можно использовать для активации этого явления. Обычно разработчик явления выбирает для действия наименование и указывает это действие в фильтре намерений для данного дявления. В оставшейся части главы мы подробно изучим фильтры намерений (intent filters).

Теперь, когда мы указали явление и связанное с ним действие, можно воспользоваться намерением для активации `BasicViewActivity`:

```
public static invokeMyApplication(Activity parentActivity)
{
   String actionName= " com.androidbook.intent.action.ShowBasicView ";
   Intent intent = new Intent(actionName);
   parentActivity.startActivity(intent);
}
```

ПРИМЕЧАНИЕ

Принято давать названия действиям в соответствии со следующим образцом: <your-packagename>.intent.action.YOUR_ACTION_NAME.

Намерения, имеющиеся в Android

Теперь у вас есть базовое представление о намерениях и мы можем опробовать их на практике, активировав одно из готовых приложений, поставляемых вместе с Android (листинг 3.32). На странице **http://developer.android.com/guide/appendix/g-app-intents.html** содержится информация о доступных приложениях и о намерениях, предназначенных для их активации. Но учтите, что этот список может изменяться в зависимости от версии Android. Он приводится здесь, чтобы помочь вам разобраться с материалом.

В набор готовых приложений должны входить следующие программы:

- приложение-браузер для просмотра веб-страниц;
- любое приложение, предназначенное для вызова телефонных номеров;

- приложение, представляющее собой номеронабиратель, в котором можно вводить номер и устанавливать соединение через пользовательский интерфейс;
- картографическое приложение для отображения карты мира с указанием координат широты и долготы;
- высокоточное картографическое приложение Google, обеспечивающее просмотр изображений улиц (street views).

Ниже приведен код, предназначенный для запуска этих приложений с помощью закрепленных за ними намерений.

Листинг 3.32. Выполнение готовых приложений Android

```
public class IntentsUtils
{
   public static void invokeWebBrowser(Activity activity)
   {
      Intent intent = new Intent(Intent.ACTION_VIEW);
      intent.setData(Uri.parse("http://www.google.com"));
      activity.startActivity(intent);
   }
   public static void invokeWebSearch(Activity activity)
   {
      Intent intent = new Intent(Intent.ACTION_WEB_SEARCH);
      intent.setData(Uri.parse("http://www.google.com"));
      activity.startActivity(intent);
   }
   public static void dial(Activity activity)
   {
      Intent intent = new Intent(Intent.ACTION_DIAL);
      activity.startActivity(intent);
   }

   public static void call(Activity activity)
   {
      Intent intent = new Intent(Intent.ACTION_CALL);
      intent.setData(Uri.parse("tel:555-555-5555"));
      activity.startActivity(intent);
   }
   public static void showMapAtLatLong(Activity activity)
   {
      Intent intent = new Intent(Intent.ACTION_VIEW);
      // geo:lat,long?z=zoomlevel&q=question-string
      intent.setData(Uri.parse("geo:0,0?z=4&q=business+near+city"));
      activity.startActivity(intent);
   }

   public static void tryOneOfThese(Activity activity)
   {
      IntentsUtils.call(activity);
   }
}
```

Чтобы выполнить этот код, вам потребуется простое явление с простым видом (такие виды были рассмотрены в предыдущем разделе), а также элемент меню для активации tryOneOfThese(activity). Создать простое меню можно так (листинг 3.33).

Листинг 3.33. Тестовое приложение для создания простого меню

```
public class HelloWorld extends Activity
{
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      TextView tv = new TextView(this);
      tv.setText("Привет, Android. Скажи Привет");
      setContentView(tv);
      registerMenu(this.getTextView());
   }
   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      super.onCreateOptionsMenu(menu);
      int base=Menu.FIRST; // value is 1
      MenuItem item1 = menu.add(base,base,base,"Test");
      return true;
   }

   @Override
   public boolean onOptionsItemSelected(MenuItem item) {
      if (item.getItemId() == 1) {
      IntentUtils.tryOneOfThese(this);
      }
      else {
         return super.onOptionsItemSelected(item);
      }
      return true;
   }
}
```

ПРИМЕЧАНИЕ

В главе 2 описано, как создать проект Android на основе этих файлов, а также как скомпилировать его и запустить. Кроме того, можете почитать начало главы 5, в которой приведены дополнительные примеры кода, описывающие работу с меню.

Намерения и универсальные идентификаторы ресурсов данных

Мы изучили простейшие намерения, для работы с которыми нам нужно всего лишь дать название действию. Явление ACTION_DIAL из листинга 3.32 представляет собой одно из таких простых действий. Для активации номеронабирателя нам требуется активировать только соответствующее действие:

```
public static void dial(Activity activity)
{
```

```
    Intent intent = new Intent(Intent.ACTION_DIAL);
    activity.startActivity(intent);
}
```

В отличие от ACTION_DIAL, намерение ACTION_CALL используется для установления вызова на указанный номер телефона и поэтому принимает дополнительный параметр Data. Этот параметр указывает на URI, который, в свою очередь, указывает на телефонный номер:

```
public static void call(Activity activity)
{
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:555-555-5555"));
    activity.startActivity(intent);
}
```

Действующая часть намерения представляет собой строку или строковую константу, которой обычно в виде префикса предшествует название пакета Java. Часть намерения, в которой содержатся данные, всегда является строкой, представляющей собой URI. Формат этого URI может быть специфичен для каждого явления, активируемого данным действием. В таком случае действие CALL решает, какой именно вид URI с данными следует ожидать. Из URI извлекается телефонный номер.

ПРИМЕЧАНИЕ

Активированное явление также может использовать URI для указания на источник данных и использовать данные из этого источника. Так происходит при работе с медиафайлами, в частности с аудио, видео и изображениями.

Обобщенные действия

Действия Intent.ACTION_CALL и Intent.ACTION_DIAL легко могут вызвать у вас ложное предположение о том, что существует взаимно однозначное соответствие между действием и элементом, который оно активирует. Для опровержения этого предположения рассмотрим противоположный пример из кода IntentUtils, содержащегося в листинге 3.32:

```
public static void invokeWebBrowser(Activity activity)
{
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse("http://www.google.com"));
    activity.startActivity(intent);
}
```

Обратите внимание: действие называется ACTION_VIEW. Как Android распознает, какое именно явление активировать в ответ на действие с таким обобщенным названием? В таких случаях Android в большей степени опирается на строение URI. Android анализирует схему URI, который начинается с http, и направляет запрос ко всем зарегистрированным явлениям, чтобы определить, какие из них понимают данную схему. Исходя из этого, он узнает, какие явления могут обработать VIEW, а затем активирует нужное явление. Чтобы этот механизм работал, в явлении-

браузере должно быть зарегистрировано намерение VIEW, которое ассоциировано со схемой данных, начинающейся с http. Объявление такого намерения в файле описания может выглядеть следующим образом:

```
<activity.....>
   <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <data android:scheme="http"/>
      <data android:scheme="https"/>
   </intent-filter>
</activity>
```

Узнать дополнительную информацию о функциях работы с данными вы можете из XML-описания элемента data, которое содержится по адресу **http://code.google.com/android/reference/android/R.styleable.html#AndroidManifestData**. К дочерним элементам или атрибутам этого узла данных XML относятся следующие:

```
host
mimeType
path
pathPattern
pathPrefix
port
scheme
```

Атрибут mimeType — один из наиболее используемых. Например, следующий фильтр намерений указывает тип MIME, соответствующий каталогу записей, и работает с явлением, которое отображает набор записей:

```
<intent-filter>
   <action android:name="android.intent.action.VIEW" />
   <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

Экран, на котором отображается отдельно взятая запись, объявляет применяемый с ней фильтр намерений, указывая тип MIME, который соответствует одиночной записи:

```
<intent-filter>
 <action android:name="android.intent.action.VIEW" />
 <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

Использование дополнительной информации

Кроме основных атрибутов действий и данных, намерение может включать и другие атрибуты, называемые *дополнительными* (extras). Дополнительный атрибут может сообщать еще какую-нибудь информацию о компоненте, получающем намерение. Дополнительные данные даются в форме пар «ключ — значение»: ключ должен начинаться с названия пакета, а значение может относиться к любому из основных типов данных или быть произвольным объектом, если только в этом

объекте используется интерфейс android.os.Parcelable. Такая дополнительная информация представлена в специальном классе Android, называемом android.os.Bundle.

Следующие два метода класса Intent открывают доступ к комплекту данных (bundle):

```
// получение комплекта данных из намерения
Bundle extraBundle = intent.getExtras();

// размещение комплекта данных в намерении
Bundle anotherBundle = new Bundle();

// заполнение комплекта данных парами "ключ – значение"


// определение комплекта данных для намерения
intent.putExtras(anotherBundle);
```

GetExtras построен прямолинейно: он возвращает комплект данных, имеющийся у намерения. PutExtras проверяет, есть ли уже у намерения комплект данных. Если это так, putExtras передает дополнительные ключи и значения из нового комплекта данных в существующий. Если комплект данных не существует, putExtras создаст его и скопирует пары «ключ — значение» из нового комплекта данных в только что созданный.

ПРИМЕЧАНИЕ

PutExtras реплицирует входящий комплект данных, а не просто указывает на него. То есть, если вы собирались изменить входящий комплект данных, вы не измените тот комплект, который находится внутри намерения.

Чтобы добавлять в дополнительный комплект данных данные основных типов, применяется ряд методов. Ниже перечислены некоторые методы, добавляющие в дополнительные пакеты данные простых типов:

```
putExtra(String name, boolean value);
putExtra(String name, int value);
putExtra(String name, double value);
putExtra(String name, String value);
```

А вот некоторые не столь простые дополнительные данные:

```
// поддержка простых массивов
putExtra(String name, int[] values);
putExtra(String name, float[] values);

// сериализуемые объекты
putExtra(String name, Serializable value);

// поддержка синтаксического разбора
putExtra(String name, Parcelable value);

// добавление дополнительного комплекта данных к указанному ключу
```

```
// дополнительные комплекты в других дополнительных комплектах
putExtra(String name, Bundle value);

// добавление комплектов данных из другого намерения
// копирование комплектов дополнительных данных
putExtra(String name, Intent anotherIntent);

// поддержка явного списка массивов
putIntegerArrayListExtra(String name, ArrayList arrayList);
putParcelableArrayListExtra(String name, ArrayList arrayList);
putStringArrayListExtra(String name, ArrayList arrayList);
```

На стороне получателя аналогичные методы, начинающиеся с get, берут информацию из дополнительных пакетов, основываясь на именах ключей.

В классе Intent определяются дополнительные ключевые строки, сопровождающие конкретные действия. Такие ключи-константы для работы с дополнительной информацией подробно описаны по адресу **http://code.google.com/android/reference/android/content/Intent.html#EXTRA_ALARM_COUNT**.

Рассмотрим два пакета с дополнительной информацией, которые используются при отправке электронной почты.

- EXTRA_EMAIL — этот строковый ключ применяется для хранения набора адресов электронной почты. Значение ключа — android.intent.extra.EMAIL. Он должен указывать на строковый массив текстовых адресов электронной почты.
- EXTRA_SUBJECT — этот ключ используется для хранения темы электронного сообщения. Значение ключа — android.intent.extra.SUBJECT. Ключ должен указывать на строку subject.

Использование компонентов для непосредственного инициирования явления

Мы рассмотрели несколько способов того, как запустить явление при помощи намерения. В частности, был показан запуск явления при помощи явного действия и запуск явления при помощи общего действия, при котором используется URI. В Android также предусмотрен более прямолинейный способ запуска явления: можно указать параметр ComponentName. Он относится к явлению и представляет собой абстракцию, объединяющую название пакета, в котором находится объект, и имя класса. В классе Intent предусмотрено несколько методов для указания компонента:

```
setComponent(ComponentName name);
setClassName(String packageName, String classNameInThatPackage);
setClassName(Context context, String classNameInThatContext);
setClass(Context context, Class classObjectInThatContext);
```

Наконец, для всех этих методов предусмотрен общий сокращенный вариант:

```
setComponent(ComponentName name);
```

ComponentName объединяет имя пакета и имя класса. Например, следующий код активирует явление contacts, поставляемое вместе с эмулятором:

```
Intent intent = new Intent();
intent.setComponent(new ComponentName(
   "com.android.contacts"
   ,"com.android.contacts.DialContactsEntryActivity");
startActivity(intent)
```

Обратите внимание — имя класса и имя пакета являются полными и используются по очереди для составления имени ComponentName до того, как передать его классу Intent.

Можно также применять непосредственно имя класса, не строя ComponentName. Снова рассмотрим фрагмент кода BasicViewActivity:

```
public class BasicViewActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.some-view);
   }
} // eof-class
```

В таком случае можно использовать для запуска данного явления следующий код:

```
Intent directIntent = new Intent(activity, BasicViewActivity.class);
activity.start(directIntent);
```

Но если для запуска явления вам требуется какое-либо намерение, необходимо зарегистрировать данное явление в файле Android.Manifest.xml следующим образом:

```
<activity android:name="BasicViewActivity"
   android:label="Test Activity">
```

При запуске явления непосредственно по имени класса или имени компонента отпадает необходимость использования фильтров намерений.

Лучшие методы разработки компонентов

Если внимательно рассмотреть конструкцию приложения Contacts, используемого в Android, можно заметить несколько общих принципов, применяемых при разработке с использованием намерений. Чтобы клиенты приложения знали намерения, необходимые для работы с ним, эти намерения определяются в приложении Contacts в трех классах, находящихся в пакете android.provider.contacts. Это следующие классы:

```
contacts.Intents
contacts.Intents.Insert // вложенный класс
contacts.Intents.UI // вложенный класс
```

В классе верхнего уровня contacts.Intents определяются основные намерения, на которые будет отвечать приложение, и события, которые приложение будет создавать по мере работы.

Во вложенном классе contacts.Intents.Insert определяются намерения поддержки и другие константы, предназначенные для вставки новых записей. Во вложенном классе contacts.Intents.UI определяются способы, позволяющие активировать пользовательский интерфейс. Вместе с намерениями также дается дополнительная информация, необходимая для их активации, в том числе имена ключей и предполагаемые типы их значений.

При разработке собственного поставщика содержимого и явлений, предназначенных для работы с этим поставщиком, вы можете последовать описанному выше принципу, создавая явные намерения и определяя константы для них в интерфейсах или классах.

Категории намерений

Явления можно классифицировать по категориям, после чего появляется возможность осуществлять поиск по имени категории. Например, при запуске Android ищет явления, имеющие категорию (тег) CATEGORY_LAUNCHER. Затем Android выбирает названия и ярлыки этих явлений, и размещает их на рабочем столе для запуска.

Рассмотрим другой пример: Android ищет явление, имеющее тег CATEGORY_HOME для того, чтобы при запуске на экране устройства отображался рабочий стол. Сходным образом CATEGORY_GADGET помечает явление как подходящее для внедрения или повторного использования в рамках другого явления.

Формат строки для категории, например CATEGORY_LAUNCHER, соответствует соглашению об определении категорий:

```
android.intent.category.LAUNCHER
```

Необходимо знать такие текстовые строки для определения категорий, поскольку явления регистрируют свои категории в файле AndroidManifest.xml в рамках определения фильтров явлений. Рассмотрим пример:

```
<activity android:name=".HelloWorld"
      android:label="@string/app_name">
   <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

ПРИМЕЧАНИЕ

Явления могут иметь определенные характерные черты, ограничивающие их или дающие им дополнительные возможности, в частности, сможете ли вы встроить такое явление в родительское явление. Такие типы характеристик явлений определяются в категориях.

Рассмотрим некоторые заданные категории Android и способы их использования (табл. 3.3).

Таблица 3.3. Категории явлений и их описание

Название категории	Описание
CATEGORY_DEFAULT	Явление может объявляться как заданное по умолчанию (DEFAULT) для работы с определенным аспектом данных, в частности типом, схемой и т. д.
CATEGORY_BROWSABLE	Явление может объявляться как доступное для просмотра (BROWSABLE). Это гарантирует, что запуск данного явления не представляет опасности для браузера
CATEGORY_TAB	Явление такого типа может встраиваться в родительское явление, содержащее вкладки (tabbed parent activity)
CATEGORY_ALTERNATIVE	Явление может объявляться как альтернативное (ALTERNATIVE) для определенного типа просматриваемых данных. Обычно такие элементы представляют собой фрагменты меню с параметрами и используются при просмотре документа. Например, вид документа для вывода на печать (print view) считается альтернативой для обычного вида (regular view)
CATEGORY_SELECTED_ALTERNATIVE	Явление может объявляться как альтернативное (ALTERNATIVE) для определенного типа данных. Это похоже на перечисление ряда редакторов, которые можно использовать для работы с текстовым или HTML-документом
CATEGORY_LAUNCHER	При присвоении явлению этой категории оно вносится в список, отображаемый на экране запуска (launcher screen)
CATEGORY_HOME	Явление такого типа — это рабочий стол. Обычно в системе предусмотрено только одно явление такого типа. Если их больше, система подскажет вам, что следует выбрать одно из них
CATEGORY_PREFERENCE	Данная категория делает явление глобальным, оно начинает отображаться на экране глобальных настроек (preferences screen)
CATEGORY_GADGET	Явление такого типа может встраиваться в родительское явление
CATEGORY_TEST	Тестовое явление
CATEGORY_EMBED	Данная категория использовалась в ранних версиях и была заменена на CATEGORY_GADGET, но она сохранена в системе для обеспечения обратной совместимости

Данные категории явлений подробно описаны по следующему адресу, относящемуся к инструментарию разработчика в Android, в разделе о классе Intent: **http://code.google.com/android/reference/android/content/Intent.html#CATEGORY_ALTERNATIVE**.

Если вы используете намерение для запуска явления, то можете задать вид явления, которое нужно выполнять. Для этого укажите его категорию. Ниже приведен пример получения информации о наборе основных явлений, категории которых совпадают с CATEGORY_SAMPLE_CODE:

```
Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
mainIntent.addCategory(Intent.CATEGORY_SAMPLE_CODE);
PackageManager pm = getPackageManager();
List<ResolveInfo> list = pm.queryIntentActivities(mainIntent, 0);
```

PackageManager — это ключевой класс, позволяющий находить явления, соответствующие определенным намерениям, не запуская эти явления. Можно просмотреть полученные явления при помощи цикла и запустить нужные из них с учетом интерфейса прикладного программирования ResolveInfo.

Следующий код имеет аналогичную логику, но вы также можете получить список доступных для запуска приложений, присвоив намерению категорию CATEGORY_LAUNCHER:

```
// Дай мне список доступных для запуска приложений.
Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
List mApps = getPackageManager().queryIntentActivities(mainIntent, 0);
```

Можно сделать еще лучше. Запустим явление на основе категории предыдущего намерения, CATEGORY_LAUNCHER:

```
public static void invokeAMainApp(Activity activity)
{
   Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
   mainIntent.addCategory(Intent.CATEGORY_LAUNCHER);
   activity.startActivity(mainIntent);
}
```

А если намерению будет удовлетворять более одного явления — какое явление выберет Android? Для решения этой проблемы в Android предусмотрено диалоговое окно **Complete Action Using** (Выбор подходящего явления из списка). В нем перечисляются все доступные явления, одно из которых можно выбрать для запуска.

Приведем еще один пример использования намерения для перехода на домашнюю страницу:

```
// перейти на рабочий стол
Intent mainIntent = new Intent(Intent.ACTION_MAIN, null);
mainIntent.addCategory(Intent.CATEGORY_HOME);
startActivity(mainIntent);
```

Если вы хотите заменить рабочий стол (home screen), заданный в Android по умолчанию, то можете сами задать явление, которое будет иметь категорию HOME. В таком случае в предыдущем коде можно будет указать явление для начала работы, так как в системе будет зарегистрировано более одного явления с категорией HOME:

```
// заменить рабочий стол
<intent-filter>
 <action android:value="android.intent.action.MAIN" />
 <category android:value="android.intent.category.HOME"/>
 <category android:value="android.intent.category.DEFAULT" />
</intent-filter>
```

Правила разложения намерений на их компоненты

Мы уже рассмотрели некоторые вопросы, касающиеся намерений. Напоминаем – мы поговорили о действиях, URI данных, дополнительных данных и, наконец, о категориях. Имея такие компоненты, Android использует для разрешения намерений в явлении следующий алгоритм.

На верхней ступени иерархии находится имя компонента, присвоенное намерению и занимающее исключительное положение. Если задано имя компонента, оно обязательно выполняется, при этом игнорируются все остальные аспекты или атрибуты намерения.

Затем Android ищет у намерения атрибут действия. Если в намерении указано действие, то оно должно быть учтено в целевом явлении как компонент фильтра намерений. Если не указано никаких других атрибутов, Android активирует данное явление. Если задано несколько явлений, система предлагает инструмент для выбора одного из них.

Затем Android просматривает фрагмент намерения, содержащий данные. Если в намерении имеется URI с данными, его тип извлекается при помощи `ContentProvider.getType()`, если тип не был сообщен вместе с присланным намерением. Целевое явление должно содержать в фильтре намерений указание на то, что оно может обрабатывать данные такого типа. Если URI с данными не является контент-URI или если тип данных не указан, учитывается схема URI. Целевое явление должно содержать указание на то, что оно может обрабатывать URI с таким типом схемы.

После этого Android просматривает категорию и выбирает только те явления, которые пригодны для использования вместе с этой категорией. В результате, если категория намерения указана, целевое явление должно объявить данную категорию в своем списке намерений.

Выполнение ACTION_PICK

До сих пор мы в основном работали с такими намерениями или действиями, которые обычно активируют другое явление, не ожидая получить в ответ на это результат. Теперь рассмотрим более сложное действие, которое возвращает значение после того, как будет активировано. `ACTION_PICK` – это обобщенное название для таких действий.

Принцип `ACTION_PICK` заключается в том, чтобы запустить явление, отображающее список элементов. После этого явление должно предоставлять пользователю возможность выбора элемента из этого списка. Когда пользователь выберет элемент, явление возвратит URI выбранного элемента вызывающей стороне. Таким образом, можно многократно использовать функцию UI для выбора нескольких элементов определенного типа.

Следует определить набор элементов, из которого будут выбираться элементы, используя для этого тип MIME, который будет указывать на курсор контента Android. Тип MIME такого URI должен быть построен по следующему образцу:

```
vnd.android.cursor.dir/vnd.google.note
```

Одна из функций явления — это получение данных от поставщика содержимого на базе URI. Это еще одна причина, по которой данные рекомендуется заключать в поставщике содержимого.

При работе с действиями, возвращающими данные такого рода, нельзя использовать метод startActivity(), поскольку startActivity() не возвращает результат. StartActivity() не может вернуть результат, так как этот метод открывает новое явление в виде модального диалогового окна в отдельном потоке, а основной поток оставляет свободным для служебных событий. Другими словами, startActivity() — это асинхронный вызов, с которым не применяются обратные вызовы и, следовательно, невозможно узнать, что произошло с активированным действием. Но если вы хотите вернуть данные, то можете использовать вариант startActivity(), называемый startActivityForResult() и приспособленный для работы с обратными вызовами.

Рассмотрим схему метода startActivityForResult() из класса Activity:

```
public void startActivityForResult(Intent intent, int requestCode)
```

Этот метод запускает явление, результат выполнения которого вы хотите получить. После завершения данного явления метод onActivityResult() исходного явления будет вызван при помощи заданного requestCode. Схема данного метода обратного вызова такова:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

Код requestCode передается методу startActivityForResult(). В качестве resultCode может использоваться RESULT_OK, RESULT_CANCELED или код, задаваемый пользователем (custom code). Пользовательские коды должны начинаться с RESULT_FIRST_USER. Параметр Intent содержит любые дополнительные данные, которые может вернуть запущенное явление. В случае с ACTION_PICK, данные, возвращаемые с намерением, указывают на URI с данными отдельно взятого элемента (листинг 3.34).

Листинг 3.34. Возвращение данных после активации явления

```
public static void invokePick(Activity activity)
{
   Intent pickIntent = new Intent(Intent.ACTION_PICK);
   int requestCode = 1;
   pickIntent.setData(Uri.parse(
      "content://com.google.provider.NotePad/notes"));
   activity.startActivityForResult(pickIntent, requestCode);
}

protected void onActivityResult(int requestCode
      ,int resultCode
      ,Intent outputIntent)
{
   super.onActivityResult(requestCode, resultCode, outputIntent);
   parseResult(this, requestCode, resultCode, outputIntent);
}
public static void parseResult(Activity activity
      , int requestCode
```

```
      , int resultCode
      , Intent outputIntent)
{
      if (requestCode != 1)
      {
         Log.d("Test", "Some one else called this, not us");
               return;
      }
      if (resultCode != Activity.RESULT_OK)
      {
         Log.d("Result code is not ok:" + resultCode);
               return;
      }
      Log.d("Test", "Result code is ok:" + resultCode);
      Uri selectedUri = outputIntent.getData();
      Log.d("Test", "The output uri:" + selectedUri.toString());

      // перейти к отображению записи
      outputIntent.setAction(Intent.VIEW);
      startActivity(outputIntent);
}
```

Константы RESULT_OK, RESULT_CANCEL и RESULT_FIRST_USER определяются в классе Activity. Числовые значения этих констант таковы:

```
RESULT_OK = -1;
RESULT_CANCEL = 0;
RESULT_FIRST_USER = 1;
```

Чтобы механизм функционировал, у разработчика должен быть код, приспособленный для выполнения требований PICK. Посмотрим, как это делается в приложении NotePad, используемом в Google. Когда элемент выбирается из списка, намерение, активировавшее явление, проверяется на предмет того, является ли оно намерением PICK. Если это так, в новом намерении задается URI данных, который затем возвращается посредством setResult():

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
 Uri uri = ContentUris.withAppendedId(getIntent().getData(), id);

 String action = getIntent().getAction();
 if (Intent.ACTION_PICK.equals(action) ||
             Intent.ACTION_GET_CONTENT.equals(action))
 {
     // Вызывающий оператор ожидает возвращения записи, вызванной
     // пользователем. Возвращается та запись,
     // на которой щелкнул пользователь.
     setResult(RESULT_OK, new Intent().setData(uri));
 } else {
     // Запуск явления для просмотра/правки выбранного элемента.
     startActivity(new Intent(Intent.ACTION_EDIT, uri));
 }
}
```

Выполнение действия GET_CONTENT

Действие ACTION_GET_CONTENT похоже на ACTION_PICK. При работе с ACTION_PICK задается URI, указывающий на набор элементов, например на коллекцию записей. Действие ACTION_PICK должно выбрать одну из записей и вернуть ее вызывающему оператору. В случае с ACTION_GET_CONTENT вы сообщаете Android, что вам требуется элемент, относящийся к конкретному типу MIME. Android ищет явления, которые могут либо создать необходимый элемент, либо выбрать элемент из имеющегося набора, чей тип MIME совпадает с заданным.

При помощи ACTION_GET_CONTENT можно выбрать запись из коллекции, поддерживаемой приложением NotePad. Для этого будет использоваться следующий код:

```
public static void invokeGetContent(Activity activity)
{
   Intent pickIntent = new Intent(Intent.ACTION_GET_CONTENT);
   int requestCode = 2;
   pickIntent.setType("vnd.android.cursor.item/vnd.google.note");
   activity.startActivityForResult(pickIntent, requestCode);
}
```

Обратите внимание, как тип намерения приравнивается к типу MIME отдельно взятой записи. В следующем фрагменте кода, где применяется код ACTION_PICK, ситуация обстоит иначе, так как вводимая информация представляет собой URI с данными:

```
public static void invokePick(Activity activity)
{
   Intent pickIntent = new Intent(Intent.ACTION_PICK);
   int requestCode = 1;
   pickIntent.setData(Uri.parse(
      "content://com.google.provider.NotePad/notes"));
   activity.startActivityForResult(pickIntent, requestCode);
}
```

Для явления, которое должно отвечать на ACTION_GET_CONTENT, необходимо зарегистрировать фильтр намерений, указывающий, что явление может предоставить элемент заданного типа MIME. Ниже показано, как эта задача решается в приложении NotePad, входящем в состав инструментария для разработки программ:

```
<activity android:name="NotesList" android:label="@string/title_notes_list">
......
<intent-filter>
   <action android:name="android.intent.action.GET_CONTENT" />
   <category android:name="android.intent.category.DEFAULT" />
   <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
      </intent-filter>
.....
</activity>
```

Оставшаяся часть кода, предназначенная для отклика на onActivityResult(), идентична соответствующему коду из предыдущего примера с ACTION_PICK. Если

несколько явлений могут возвращать одинаковый тип MIME, Android отобразит диалоговое окно с вариантами на выбор, где вы сможете указать одно из явлений. Однако по умолчанию может быть жестко задан один из вариантов, и выбрать другой заголовок вы не сможете. Чтобы обойти это ограничение, в Android был введен метод createChooser класса Intent, позволяющий использовать специальный оператор выбора, чей заголовок всегда можно изменить. Ниже показано, как активировать такой оператор:

```
// начать с целевого типа Intent, на который необходимо указать
Intent intent = new Intent();
intent.setType(…);
Intent chooserIntent = Intent.createChooser(intent, "Hello use this title");
activity.startActivityForResult(chooserIntent);
```

Дополнительные ресурсы для углубленного изучения материала данной главы

В данном разделе приводится небольшой список полезных ссылок, которые помогут вам закрепить и углубить знания, приобретенные в этой главе.

По этой ссылке можно найти наиболее актуальный список ресурсов, которые поддерживаются в Android: http://developer.android.com/guide/topics/resources/availableresources.html.

По этому адресу рассматриваются вопросы, связанные с локализацией и использованием ресурсов для данной цели: http://developer.android.com/guide/topics/resources/resources-i18n.html.

Этот API для работы с ресурсами удобен при явном поиске ресурсов: http://developer.android.com/reference/android/content/res/Resources.html.

По следующему адресу рассказано о документации Android, описывающей поставщиков содержимого: http://developer.android.com/guide/topics/providers/content-providers.html.

Здесь описываются API для поставщиков содержимого. О принципах работы с поставщиками содержимого рассказано по следующей ссылке: http://developer.android.com/reference/android/content/ContentProvider.html.

По этому адресу понятно рассказано о том, что такое UriMatcher: http://developer.android.com/reference/android/content/UriMatcher.html.

Следующая ссылка помогает считывать данные непосредственно из поставщика содержимого или из базы данных: http://developer.android.com/reference/android/database/Cursor.html.

Это домашняя страница базы данных SQLite: http://www.sqlite.org/sqlite.html.

Здесь приводится обзор намерений, используемых в Android: http://developer.android.com/reference/android/content/Intent.html.

По этой ссылке находится список намерений, предназначенных для активации приложений Google: http://developer.android.com/guide/appendix/g-app-intents.html.

Здесь есть некоторая информация, которая пригодится вам при регистрации фильтров намерений: http://developer.android.com/reference/android/content/IntentFilter.html.

Создатели данной страницы попытались собрать коллекцию открытых намерений от самых разных производителей: http://www.openintents.org/.

Резюме

В этой главе мы рассмотрели три ключевых элемента, входящих в состав инструментария для разработки в Android: ресурсы, поставщики содержимого и намерения.

В разделе о ресурсах вы узнали, как создавать ресурсы в XML-файлах, а затем использовать их ID при программировании.

В разделе о поставщиках содержимого вы изучили, как работать с URI и типами MIME, а также как заключать в поставщике содержимого средства для доступа к данным. Мы также в общих чертах обсудили вопросы, связанные с созданием и использованием базы данных SQLite, которая должна работать хорошо даже без применения поставщиков содержимого.

В третьем разделе было показано, как использовать намерения для запуска других явлений несколькими способами. Теперь вы знаете, как намерения помогают проложить путь к методу «подключи и работай» (plug and play) и обеспечить многократное применение элементов на уровне пользовательского интерфейса. Хорошо усвоив эти три базовые концепции, вы будете гораздо лучше понимать принципы использования Android SDK и программирование пользовательских интерфейсов в Android.

4 Создание пользовательских интерфейсов и использование элементов управления

К настоящему моменту вы уже знакомы с основами Android, но еще не работали с пользовательским интерфейсом (user interface, UI). В этой главе рассмотрим пользовательские интерфейсы и их элементы управления. Мы начнем с общей философии разработки пользовательских интерфейсов в Android, а затем опишем наиболее типичные элементы управления, входящие в состав Android SDK. Кроме того, рассмотрим диспетчеры шаблонов (layout managers) и адаптеры видов (view adapters). В заключение познакомимся с модулем для просмотра иерархии — инструмента, используемого в Android для отладки и оптимизации пользовательских интерфейсов.

Разработка пользовательских интерфейсов в Android

Разрабатывать пользовательские интерфейсы в Android весело. Весело потому, что некоторые неприятные свойства других платформ, касающиеся этого вида разработки, в Android удалось устранить. Например, Swing должен поддерживать как приложения для локального компьютера, так и апплеты Java. В базовых классах Java (JFC) содержится масса функций, многими из которых не хочется пользоваться и очень нелегко управлять. Другой пример — JavaServer Faces (JSF). Фреймворк JSF часто используется для разработки веб-приложений, он основан на технологии «серверные страницы Java» (JSP) и сервлетах. Поэтому, прежде чем начать работать с JSF, необходимо изучить все фреймворки, на которых построена эта технология.

Приятно, что в Android, в отличие от других платформ, такой балласт отсутствует. В Android мы имеем простой фреймворк с небольшим набором элементов управления, поставляемых вместе с системой. Доступная площадь экрана обычно ограниченна. Эти моменты в сочетании с тем фактом, что обычно пользователь собирается выполнять только одно конкретное действие, позволяют с легкостью построить хороший пользовательский интерфейс, с которым будет удобно работать.

В состав Android SDK при поставке входит набор элементов управления, которые можно задействовать в создании пользовательского интерфейса приложения. Как и в других подобных инструментах, в Android SDK имеются поля для ввода текста (text fields), кнопки (buttons), списки (lists), таблицы (grids) и т. д. Кроме того, в Android SDK есть набор элементов управления, специально предназначенных для работы с мобильными устройствами.

Все наиболее распространенные элементы управления работают на базе двух классов: `android.view.View` и `android.view.ViewGroup`. По названию первого класса можно догадаться, что в классе `View` представлен объект `View` общего назначения. Типичные элементы управления в Android в конечном счете дополняют класс `View`. `ViewGroup` — это тоже вид, но такой, в котором содержатся другие виды. `ViewGroup` является базовым классом для ряда классов шаблонов (layout classes). В Android, как и в Swing, используется концепция «шаблон», позволяющая управлять тем, как элементы управления располагаются друг относительно друга внутри содержащего их вида (container view). В дальнейшем вы увидите, что при работе с вариантами шаблонов совсем просто контролировать положение и ориентацию элементов управления в пользовательских интерфейсах.

В Android применяется несколько подходов к созданию пользовательских интерфейсов. Можно целиком создать такой интерфейс в коде. Кроме того, пользовательские интерфейсы можно определять в XML. Можно даже объединить оба подхода — определить пользовательский интерфейс в XML, а потом ставить на него ссылки в коде и при помощи кода его изменять. Чтобы продемонстрировать эти три подхода, мы рассмотрим, как создать простой пользовательский интерфейс при помощи каждого из них.

Прежде чем приступить к работе, разберемся с терминологией. В этой книге, а также в других книгах по Android вам встретятся термины *вид* (view), *элемент управления* (control), *виджет* (widget), *контейнер* (container) и *шаблон* (layout), которые касаются разработки пользовательских интерфейсов. Если вы ранее не занимались программированием для Android или вообще не сталкивались с разработкой пользовательских интерфейсов, эти термины могут быть вам не знакомы. Мы привели в табл. 4.1 краткое описание каждого из них.

Таблица 4.1. Терминология, связанная с пользовательскими интерфейсами

Термин	Описание
Вид, виджет, элемент управления	Все это элементы пользовательского интерфейса. К их числу относятся кнопка, координатная сетка, список, окно, диалоговое окно и т. д. Термины «вид» «виджет» и «элемет управления» в этой главе являются взаимозаменяемыми
Контейнер	Это вид, в котором содержатся другие виды. Например, координатная сетка может считаться контейнером, так как содержит ячейки, каждая из которых является видом
Шаблон	Это XML-файл, используемый для описания вида

На рис. 4.1 показан скриншот приложения, которое нам предстоит создать. Рядом со скриншотом — схема иерархии шаблонов элементов управления и контейнеров в приложении.





Рис. 4.1. Пользовательский интерфейс и шаблон явления

Мы будем обращаться к данной иерархии шаблонов при обсуждении примеров программ. Пока будем помнить, что в приложении есть одно явление. Пользовательский интерфейс явления состоит из трех контейнеров: контейнера, содержащего имя, контейнера, содержащего адрес, и внешнего контейнера, включающего в себя два упомянутых выше дочерних контейнера.

В листинге 4.1 показано, как полностью построить пользовательский интерфейс в коде. Начнем с создания нового проекта Android с явлением `MainActivity`, а затем скопируем код из листинга 4.1 в наш класс `MainActivity`.

Листинг 4.1. Создание простого пользовательского интерфейса исключительно в коде

```
package pro.android;
import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup.LayoutParams;
import android.widget.LinearLayout;
import android.widget.TextView;
public class MainActivity extends Activity
{
   private LinearLayout nameContainer;

   private LinearLayout addressContainer;

   private LinearLayout parentContainer;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);

      createNameContainer();

      createAddressContainer();

      createParentContainer();

      setContentView(parentContainer);

   }

   private void createNameContainer()
```

```
{
   nameContainer = new LinearLayout(this);

   nameContainer.setLayoutParams(new LayoutParams
         (LayoutParams.FILL_PARENT,LayoutParams.WRAP_CONTENT));
   nameContainer.setOrientation(LinearLayout.HORIZONTAL);

   TextView nameLbl = new TextView(this);

   nameLbl.setText("Имя: ");
   nameContainer.addView(nameLbl);

   TextView nameValueLbl = new TextView(this);
   nameValueLbl.setText("John Doe");

   nameContainer.addView(nameValueLbl);
}

private void createAddressContainer()
{
   addressContainer = new LinearLayout(this);

   addressContainer.setLayoutParams(new LayoutParams
         (LayoutParams.FILL_PARENT,LayoutParams.WRAP_CONTENT));
   addressContainer.setOrientation(LinearLayout.VERTICAL);

   TextView addrLbl = new TextView(this);

   addrLbl.setText("Address:");

   TextView addrValueLbl = new TextView(this);

   addrValueLbl.setText("911 Hollywood Blvd");

   addressContainer.addView(addrLbl);
   addressContainer.addView(addrValueLbl);
}

private void createParentContainer()
{
   parentContainer = new LinearLayout(this);

   parentContainer.setLayoutParams(new LayoutParams
         (LayoutParams.FILL_PARENT,LayoutParams.FILL_PARENT));
   parentContainer.setOrientation(LinearLayout.VERTICAL);

   parentContainer.addView(nameContainer);
   parentContainer.addView(addressContainer);
}
```

Как показано в листинге 4.1, в явлении находится три объекта LinearLayout. Выше мы упоминали о том, что в объектах шаблонов содержится логика для размещения нескольких объектов на участке экрана. Например, LinearLayout отвечает за вертикальное или горизонтальное расположение элементов управления. В объектах шаблонов могут содержаться любые виды — в том числе объекты шаблонов.

В объекте nameContainer находятся два элемента управления TextView: один из них — это обозначение Name:, а в другом содержится само имя (например, John Doe). В addressContainer также располагаются два элемента управления TextView. Разница между двумя контейнерами заключается в том, что nameContainer задает расположение по горизонтали, а addressContainer — по вертикали. Оба этих контейнера находятся внутри parentContainer, являющегося основным видом (root view) данного явления. После того как контейнеры будут готовы, явление определяет содержимое этого вида как содержимое root view, вызывая для этого setContentView(parentContainer). Когда мы переходим к отображению пользовательского интерфейса явления, сначала вызывается основной вид. Затем основной вид вызывает свои дочерние виды, чтобы они также отобразились, а дочерние виды вызывают содержащиеся в них элементы управления и т. д., пока не отобразится весь пользовательский интерфейс.

В листинге 4.1 показано несколько элементов управления LinearLayout. На практике два из них располагаются по вертикали, а один — по горизонтали. Это означает, что два вида находятся рядом друг с другом по горизонтали. AddressContainer расположен по вертикали. Это означает, что один элемент управления TextView будет «лежать» на другом. ParentContainer также располагается по вертикали, именно поэтому nameContainer оказывается выше, чем addressContainer. Следует отметить тонкую разницу между двумя контейнерами, расположенными по вертикали, addressContainer и parentContainer: parentContainer занимает всю ширину и высоту экрана.

```
parentContainer.setLayoutParams(new LayoutParams
        (LayoutParams.FILL_PARENT,LayoutParams.FILL_PARENT));
```

А addressContainer охватывает его содержимое по вертикали:

```
addressContainer.setLayoutParams(new LayoutParams
        (LayoutParams.FILL_PARENT,LayoutParams.WRAP_CONTENT));
```

Теперь создадим такой же пользовательский интерфейс в XML (листинг 4.2). Как вы помните из главы 3, XML-файлы шаблонов хранятся в каталоге ресурсов (/res/), в папке, называемой layout. Чтобы выполнить этот пример на практике, создадим в Eclipse новый проект Android. По умолчанию у нас получится XML-шаблон с именем main.xml, расположенный в каталоге res/layout. Дважды щелкните на main.xml, чтобы просмотреть его содержимое. Eclipse отобразит визуальный редактор для работы с файлом шаблона. Наверное, в верхней части этого вида будет находиться строка Hello World, MainActivity! или подобная. Щелкните на вкладке **main.xml** в нижней части вида, чтобы просмотреть XML-код файла main.xml. В этом коде вы увидите элементы управления LinearLayout и TextView. При помощи вкладок **Layout** или **main.xml** или обеих сразу перепишите листинг 4.2 для файла main.xml. Сохраните его.

Листинг 4.2. Создание пользовательского интерфейса целиком в XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <!-- КОНТЕЙНЕР ИМЕНИ -->
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

         <TextView android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Имя:" />

         <TextView android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="John Doe" />

   </LinearLayout>

   <!-- КОНТЕЙНЕР АДРЕСА -->
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

         <TextView android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:text="Адрес:" />

         <TextView android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:text=
                                     "911 Hollywood Blvd." />
   </LinearLayout>

</LinearLayout>
```

В каталоге `src` нашего нового проекта содержится задаваемый по умолчанию файл JAVA, в котором определяется класс `Activity`. Дважды щелкните на этом файле, чтобы просмотреть его содержимое. Обратите внимание на инструкцию `setContentView(R.layout.main)`. Фрагмент XML, показанный в листинге 4.2, в комбинации с вызовом `setContentView(R.layout.main)`, позволяет отобразить тот же пользовательский интерфейс, который мы целиком сгенерировали в коде. Файл XML должен быть понятен без пояснений, но обратите внимание на то, что в нем мы определили три вида. Первый — `LinearLayout` — это эквивалент родительского контейнера. Чтобы данный контейнер получил вертикальную ориентацию, соответствующее свойство необходимо задать следующим образом: `android:orientation="vertical"`. В родительском контейнере будут содержаться два контейнера `LinearLayout`, а именно `nameContainer` и `addressContainer`.

Пример из листинга 4.2 является несколько надуманным. В частности, не имеет смысла жестко кодировать значения элементов управления `TextView` в XML. В идеальном случае пользовательские интерфейсы следует создавать в XML, а затем ставить ссылки из кода на элементы управления. Такой подход позволяет

связывать динамические данные с элементами управления, созданными на этапе проектирования. Рекомендуется использовать именно такой подход.

В листинге 4.3 показан тот же пользовательский интерфейс с немного измененным XML. В этом XML элементам управления TextView присваиваются идентификаторы (ID), на которые затем можно ссылаться в коде.

Листинг 4.3. Создание пользовательского интерфейса в XML с применением ID

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <!-- КОНТЕЙНЕР ИМЕНИ -->
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

         <TextView android:id="@+id/nameText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@+string/name_text" />

         <TextView android:id="@+id/nameValueText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

   </LinearLayout>

   <!-- КОНТЕЙНЕР АДРЕСА -->
   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

         <TextView android:id="@+id/addrText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@+string/addr_text" />

         <TextView android:id="@+id/addrValueText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
   </LinearLayout>

</LinearLayout>
```

Код в листинге 4.4 позволяет понять, как получить ссылки на элементы управления, определенные в файле XML, чтобы задать их свойства.

Листинг 4.4. Проставление ссылок на элементы управления в ресурсах во время исполнения

```
setContentView(R.layout.main);

TextView nameValue = (TextView)findViewById(R.id.nameValueText);
```

```
nameValue.setText("John Doe");
TextView addrValue = (TextView)findViewById(R.id.addrValueText);
addrValue.setText("911 Hollywood Blvd.");
```

Код в листинге 4.4 достаточно прост, но следует отметить, что сначала загружается ресурс (путем вызова `setContentView(R.layout.main)`) и только потом мы можем вызвать `findViewById()` — ведь у нас нет возможности поставить ссылки на виды, которые еще не загружены.

Обычные элементы управления в Android

В этом разделе мы поговорим об элементах управления, наиболее используемых в Android SDK. Начнем с текстовых элементов управления, а затем рассмотрим кнопки, флажки, радиокнопки (кнопки с зависимой фиксацией), списки, координатные сетки, элементы управления датой и временем и элемент для построения карт (map-view control). Мы также рассмотрим инструменты управления шаблонами. В конце главы мы покажем, как можно написать код для собственных элементов управления.

Текстовые элементы управления

Текстовые элементы управления — самые простые из элементов управления, которые есть в Android. В Android содержится полный, но не чрезмерный набор текстовых элементов управления. В этом разделе будут рассмотрены элементы `TextView`, `EditText`, `AutoCompleteTextView` и `MultiCompleteTextView`. На рис. 4.2 такие элементы управления показаны в действии.



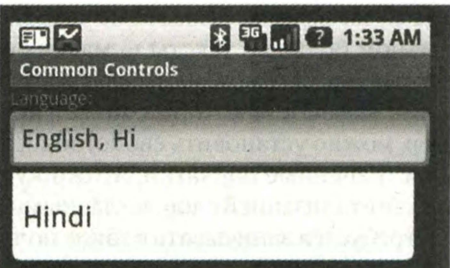


Рис. 4.2. Текстовые элементы управления в Android

TextView

Элемент TextView предназначен для отображения текста, он не позволяет редактировать текст. Из-за этого можно подумать, что данный элемент управления — пустышка. Это не так. У элемента управления TextView есть несколько интересных свойств, которые делают его очень удобным. Если вы знаете, что в содержимом TextView обязательно будет гиперссылка из Интернета, можно установить для свойства autoLink значение web — тогда элемент найдет и выделит URL. Более того, если пользователь нажмет TextView, система откроет в браузере ссылку, содержащуюся в элементе.

Более интересный случай использования TextView связан с классом android.text.util.Linkify (листинг 4.5).

Листинг 4.5. Использование класса Linkify с TextView

```
TextView tv =(TextView)this.findViewById(R.id.cctvex);
tv.setText("Пожалуйста, посетите мой сайт http://www.sayedhashimi.com
или напишите мне по адресу sayed@sayedhashimi.com.");
Linkify.addLinks(tv, Linkify.ALL);
```

Как видите, можно передать TextView классу Linkify, чтобы находить и добавлять ссылки внутри TextView. В нашем примере мы вызываем метод addLinks() класса Linkify, передавая TextView и маску, в которой указываем, ссылки каких типов должен искать Linkify.

Linkify может создавать ссылки на последовательности символов, имеющие форму телефонного номера, адреса электронной почты, веб-гиперссылки или географических координат. Передавая Linkify.ALL, мы приказываем классу «перелинковать» все эти типы ссылок. При щелчке на ссылке система вызовет намерение, по умолчанию заданное для данного действия. Например, при щелчке на веб-ссылке запустится браузер, в котором сразу же откроется эта ссылка. При щелчке на номере телефона запустится номеронабиратель и т. д. Класс Linkify может выполнять такие задачи без внесения в него дополнительных изменений. Вы также можете создать класс, который будет ссылаться на другие виды контента (например, на имена). Для этого вместе с URI поставщика содержимого нужно указать специальное регулярное выражение.

EditText

Элемент управления EditText является субклассом TextView. Как понятно из названия, элемент управления EditText позволяет редактировать текст. EditText не так многофункционален, как элементы для редактирования текста, входящие в состав JFC, но ведь пользователи устройств с Android вряд ли будут набирать на мобильном большие тексты — максимум, нужно будет напечатать пару абзацев. Следовательно, возможности класса пусть и ограничены, но отлично подходят при решении задач, для которых может использоваться мобильное устройство. Например, можно установить свойство autoText, которое позволяет исправлять самые распространенные опечатки. Можно установить свойство capitalize для управления капитализацией слов, заглавными буквами в начале предложений и т. д. Если вам потребуется записывать в такие поля телефонные номера, установите свойство phoneNumber. Можно также установить свойство password, если вам потребуется создать поле для ввода пароля.

По умолчанию элемент управления EditText должен отображать текст в одной строке и при необходимости распространять его на несколько строк. Другими словами, если текст, напечатанный пользователем, занял всю первую строку, появляется вторая строка и т. д. Но вы можете оставить для пользовательского ввода и только одну строку, задав для свойства singleLine значение true. В таком случае пользователь продолжит печатать текст на той же строке.

Все программирование для мобильных устройств направлено на то, чтобы помочь пользователю быстро принимать решения. Следовательно, часто приходится выделять или особым образом оформлять часть содержимого элемента EditText. Это можно делать статически или динамически. При статическом подходе разметка применяется напрямую к строкам ресурсов (<string name="styledText"><i>Static</i> стиль в EditText.</string>), а затем ставится ссылка на ресурс в XML-файле или из кода. Обратите внимание, со строковыми ресурсами можно использовать только следующие HTML-теги: <i>, <b> и <u>.

Для оформления содержимого элемента управления EditText посредством программирования потребуется выполнить еще немного работы, но само оформление станет значительно более гибким (листинг 4.6).

Листинг 4.6. Динамическое применение оформления к содержимому EditText

```
EditText et =(EditText)this.findViewById(R.id.cctvex5);
et.setText("Динамическое оформление содержимого editText");
Spannable spn = et.getText();
spn.setSpan(new BackgroundColorSpan(Color.RED), 0, 7,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
spn.setSpan(new StyleSpan(android.graphics.Typeface.BOLD_ITALIC)
, 0, 7, Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

Из листинга 4.6 видно, что мы можем получить содержимое EditText (например, объект Spannable), а затем установить стили для отдельных фрагментов текста. Код из этого листинга делает текст полужирным курсивом и задает для текста красный фон. Теперь вы не ограничены тремя эффектами — полужирным выделением, курсивом и нижним подчеркиванием. Можно использовать такие шрифтовые выделения, как верхний и нижний индексы, зачеркивание и др.

AutoCompleteTextView

Элемент управления AutoCompleteTextView — это вариант TextView, в котором предусмотрена функция автоматического завершения ввода. Иными словами, при вводе текста в TextView элемент управления предлагает варианты окончания вводимого текста пользователю. В листинге 4.7 показан элемент управления AutoCompleteTextView.

Листинг 4.7. Использование элемента управления AutoCompleteTextView

```
AutoCompleteTextView actv = (AutoCompleteTextView)
                        this.findViewById(R.id.ccactv);

ArrayAdapter<String> aa = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line,
new String[] {"English", "Hebrew", "Hindi", "Spanish", "German", "Greek" });

actv.setAdapter(aa);
```

Элемент управления AutoCompleteTextView, показанный в листинге 4.7, помогает пользователю при выборе языка. Например, если пользователь напишет en, система предложит вариант English. При вводе gr будет дан вариант Greek и т. д.

Если вы уже пользовались элементами управления с функцией предложения вариантов (suggestion control) или похожими на них элементами с функцией автозаполнения, то вам известно, что такие элементы управления состоят из двух частей: строки для ввода текста и элемента, в котором в виде списка предлагаются варианты. Это общая концепция. Для работы с таким элементом нужно создать сам элемент, список вариантов, сообщить этот список элементу управления и, возможно, также указать, как именно должны отображаться варианты. Можно также создать второй элемент управления, в котором отображались бы варианты, и связать вместе два элемента управления.

Как видно из листинга 4.7, в Android сделать это очень просто. Для использования AutoCompleteTextView можно определить этот элемент в файле шаблона, а затем поставить на него ссылку от явления. Затем создается класс адаптера. В нем содержатся варианты, которые будут предлагаться на выбор, а также определяется ID элемента, в котором будет отображаться тот или иной вариант (в данном случае — обычный элемент списка). В листинге 4.7 второй параметр ArrayAdapter приказывает адаптеру использовать для показа вариантов простой элемент списка. Последний этап — связать адаптер с элементом AutoCompleteTextView. Для этого используется метод setAdapter().

MultiAutoCompleteTextView

Если вы достаточно хорошо познакомились с элементом AutoCompleteTextView, то вам уже известно, что этот элемент предлагает варианты только для *законченного варианта* текста, который может быть введен в текстовое поле. Это означает, что, если вы пишете предложение, система не будет выдавать подсказки для каждого слова. Для предложения вариантов по каждому слову применяется метод MultiAutoCompleteTextView. Его можно использовать, чтобы предлагать варианты слов уже при наборе их пользователем. Например, на рис. 4.2 показано, что пользователь ввел слово English, поставил запятую и начал писать Hi, в ответ на что система предложила вариант Hindi. Если бы пользователь продолжил вводить символы, система предложила бы и другие варианты.

Работа с MultiAutoCompleteTextView напоминает использование AutoCompleteTextView. Разница заключается в том, что вам потребуется указать элементу управления, с какого места снова предлагать варианты ввода. Например, на рис. 4.2 вы видите, что элемент управления начинает предлагать варианты ввода или в начале предложения, или после запятой. Для элемента MultiAutoCompleteTextView требуется задать специальный генератор меток (tokenizer), способный производить синтаксический разбор предложения и сообщать системе, откуда снова начинать предлагать варианты ввода. В листинге 4.8 показано, как использовать элемент управления MultiAutoCompleteTextView.

Листинг 4.8. Использование элемента управления MultiAutoCompleteTextView

```
MultiAutoCompleteTextView mactv = (MultiAutoCompleteTextView) this
        .findViewById(R.id.ccmactv);
```

```
ArrayAdapter<String> aa2 = new ArrayAdapter<String>(this,
            android.R.layout.simple_dropdown_item_1line,
new String[] {"English", "Hebrew", "Hindi", "Spanish", "German", "Greek" });

mactv.setAdapter(aa2);

mactv.setTokenizer(new MultiAutoCompleteTextView.CommaTokenizer());
```

Единственное серьезное различие между листингами 4.7 и 4.8 — это использование `MultiAutoCompleteTextView` и вызов метода `setTokenizer()`. Поскольку в данном случае применяется `CommaTokenizer`, то после ввода в поле `EditText` запятой (,) в данном поле будут снова предлагаться варианты ввода, и для этого система будет пользоваться массивом строк. После ввода в строку любого другого символа система не будет снова предлагать варианты. Так, если вы даже напишете `French Spani` — за частью слова `Spani` не последует никаких предложений о вариантах, так как слово `Spani` не идет после запятой.

Элементы управления кнопки

Обычно кнопки присутствуют в любом наборе виджетов, и Android в данном случае не исключение. В Android предлагается типичный набор кнопок, а также некоторые дополнительные кнопки. В этом разделе мы рассмотрим кнопки трех типов: обычная кнопка (basic button), кнопка с изображением (image button) и кнопка-переключатель (toggle button). На рис. 4.3 показан пользовательский интерфейс, в котором есть кнопки трех этих видов. Сверху расположена обычная кнопка, в середине — кнопка с изображением, а внизу — кнопка-переключатель.

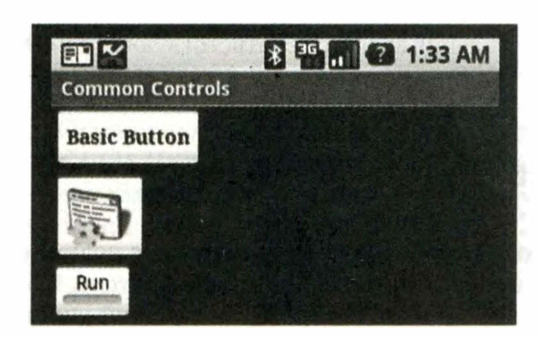


Рис. 4.3. Элементы управления — кнопки в Android

Начнем с обычной кнопки.

Элемент управления Button

Класс Android, в котором хранится информация об обычных кнопках — это `android.widget.Button`. О данном типе кнопок можно сказать не так много, кроме того что они используются для работы с событиями на нажатие (click events) (листинг 4.9).

Листинг 4.9. Обработка событий, происходящих в ответ на нажатие кнопки

```
<Button android:id="@+id/ccbtn1"
   android:text="@+string/basicBtnLabel"
   android:typeface="serif" android:textStyle="bold"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content" />

Button btn = (Button)this.findViewById(R.id.ccbtn1);
btn.setOnClickListener(new OnClickListener().
{
   public void onClick(View v)
   {

      Intent intent = getButtonIntent();
      intent.setAction("some intent data");
      setResult(RESULT_OK, intent);
      finish();
   }
});
```

В листинге 4.9 показано, как зарегистрировать событие, происходящее в ответ на нажатие кнопки. Для этого с OnClickListener вызывается метод setOnClickListener(). В листинге 4.9 мы на лету создаем анонимный процесс-слушатель для обработки событий, возникающих в ответ на btn. При щелчке на кнопке вызывается метод onClick() процесса-слушателя.

В версии Android 1.6 и выше настройка обработчика щелчков на кнопке (или кнопках) упрощена. В XML для button указывается атрибут, соответствующий следующему образцу:

```
android:onClick="myClickHandler"
```

Для него в классе явления задается соответствующий метод обработки нажатий кнопки:

```
public void myClickHandler(View target) {
   switch(target.getId()) {
   case R.id.ccbtn1:
```

При вызове метода-обработчика для него задана цель — объект View, представляющий собой ту кнопку, которую нажал пользователь. Обратите внимание, как оператор — переключатель метода обработки нажатий кнопки использует ID кнопок для выбора логики запуска процессов. Если вы применяете этот метод, вам не придется специально создавать в коде каждый отдельный объект button, и при помощи одного и того же метода вы сможете обслуживать несколько кнопок. Благодаря этому система становится понятнее, упрощается ее поддержка. Подобный метод работает и с другими типами кнопок.

Элемент управления ImageButton

Кнопки-изображения в Android содержатся в android.widget.ImageButton. Работа с такими кнопками практически не отличается от использования обычных кнопок (листинг 4.10).

Листинг 4.10. Использование ImageButton

```
<ImageButton android:id="@+id/imageBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />

ImageButton btn = (ImageButton)this.findViewById(R.id.imageBtn);
btn.setImageResource(R.drawable.icon);
```

Изображение, отображаемое на кнопке, можно устанавливать динамически, вызывая `setImageResource()` или изменяя XML-файл шаблона (задавая свойство `android:src` для ID изображения), как это показано в листинге 4.11.

Листинг 4.11. Установка изображения для ImageButton средствами XML

```
<ImageButton android:id="@+id/imageBtn"
 android:src="@drawable/btnImage"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />
```

Элемент управления ToggleButton

Элемент `ToggleButton`, подобно флажку (checkbox) или переключателю (radio button), — это кнопка, которая может оказываться в одном из двух состояний: активна (`On`) или неактивна (`Off`). По умолчанию на кнопке написано **On** (Включено), если она активна, и **Off** (Выключено) — если нет.

В листинге 4.12 приведен соответствующий пример.

Листинг 4.12. Кнопка ToggleButton в Android

```
<ToggleButton android:id="@+id/cctglBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Toggle Button"/>
```

Если надписи **On**/**Off** (Включено/Выключено) для вашей программы не подходят, их можно изменить. Например, если вы хотите запускать и останавливать при помощи кнопки-переключателя фоновый процесс, на кнопке можно написать **Run** (Запустить) и **Stop** (Остановить), воспользовавшись свойствами `android:textOn` и `android:textOff` (листинг 4.13). Поскольку надписи **On** (Включено) и **Off** (Выключено) на кнопке-переключателе являются отдельными атрибутами, атрибут `android:text` на самом деле не используется с `ToggleButton`. Этот атрибут доступен, так как он наследуется (от `TextView`), но в данном случае без него можно обойтись.

Листинг 4.13. Установка обозначения для ToggleButton

```
<ToggleButton android:id="@+id/cctglBtn"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:textOn="Run"
 android:textOff="Stop"
 android:text="Toggle Button"/>
```

Элемент управления CheckBox

Элемент управления флажок (checkbox) присутствует практически во всех наборах виджетов. Флажки поддерживаются и в HTML, и в JFC, и в JSF. Флажок — это элемент, который может переключать систему между двумя состояниями.

В Android для создания флажка необходимо сделать экземпляр android.widget.CheckBox (листинг 4.14 и рис. 4.4).

Листинг 4.14. Создание флажков

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical" android:layout_width="fill_parent"
 android:layout_height="fill_parent">

<CheckBox android:text="Chicken"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />

<CheckBox android:text="Fish"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />

<CheckBox android:text="Steak"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />

</LinearLayout>
```

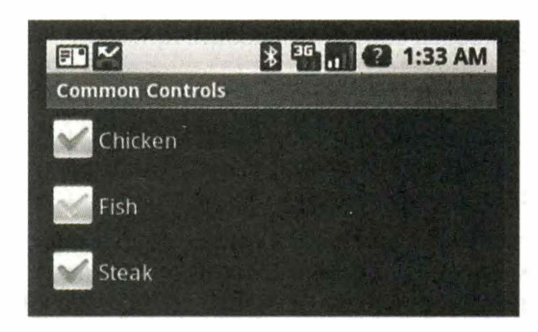


Рис. 4.4. Использование флажка

Для управления состояниями флажка используйте setChecked() или toggle(). Чтобы узнать состояние флажка, вызовите isChecked().

Если в системе потребуется задействовать специфическую логику, в соответствии с которой флажок будет включаться или выключаться, зарегистрируйте событие, соответствующее on, вызвав setOnCheckedChangeListener() с реализацией интерфейса OnCheckedChangeListener. Затем будет нужно реализовать метод onCheckedChanged(), который будет вызываться при изменении состояния флажка.

Элемент управления RadioButton

Переключатели (радиокнопка) — это неотъемлемая часть любого инструментария, предназначенного для создания пользовательских интерфейсов. Они используются в тех случаях, когда от пользователя требуется выбрать только один элемент из предлагаемого списка. Для обеспечения работы такой модели выбора единствен-

ного элемента переключатели обычно объединяются в группы, и в каждой группе может быть одновременно отмечен только один элемент.

Чтобы создать в Android группу переключателей, сначала нужно создать RadioGroup, а затем заполнить эту группу переключателями. Пример показан в листинге 4.15 и на рис. 4.5.

Листинг 4.15. Использование переключателей в Android

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

<RadioGroup  android:id="@+id/rBtnGrp"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:orientation="vertical" >

<RadioButton    android:id="@+id/chRBtn" android:text="Chicken"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

<RadioButton android:id="@+id/fishRBtn" android:text="Fish"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

<RadioButton android:id="@+id/stkRBtn" android:text="Steak"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

</RadioGroup>

</LinearLayout>
```

В Android группа переключателей внедряется в систему при помощи android.widget.RadioGroup, а для создания отдельного переключателя применяется android.widget.RadioButton.

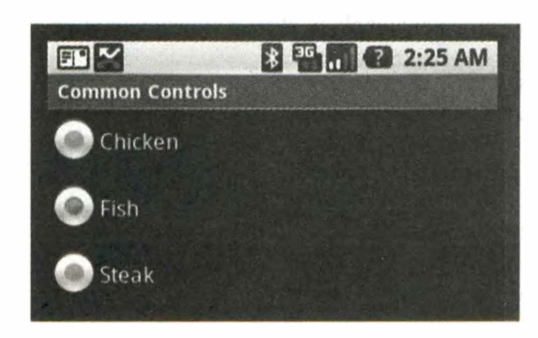


Рис. 4.5. Использование переключателя

Обратите внимание: по умолчанию переключатель не установлен ни в одно из положений, хотя в файле XML можно указать один из вариантов, который будет выбран уже при загрузке переключателя. Чтобы установить переключатель в одно из положений средствами программирования, можно получить ссылку на переключатель и вызвать setChecked():

```
RadioButton rbtn = (RadioButton)this.findViewById(R.id.stkRBtn);
rbtn.setChecked(true);
```

Можно также использовать метод toggle(), чтобы установить переключатель из одного состояния в другое. Как и при работе с CheckBox, будут использоваться уведомления о переходе положения переключателя в активное или неактивное состояние в результате определенных событий, если вы будете вызывать setOnCheckedChangeListener() с применением интерфейса OnCheckedChangeListener.

Обратите внимание: в RadioGroup могут содержаться не только переключатели, но и другие элементы. Например, в листинге 4.16 за последним положением переключателя идет TextView. Заметьте также, что один из переключателей не входит в группу, в которую входят остальные переключатели.

Листинг 4.16. Группа элементов, в которую входят не только переключатели

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

<RadioButton android:id="@+id/anotherRadBtn"
   android:text="Outside"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
<RadioGroup android:id="@+id/rdGrp"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
<RadioButton android:id="@+id/chRBtn"
   android:text="Chicken"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
<RadioButton android:id="@+id/fishRBtn"
   android:text="Fish"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
<RadioButton android:id="@+id/stkRBtn"
   android:text="Steak"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>

<TextView android:text="My Favorite"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
</RadioGroup>

</LinearLayout>
```

В листинге 4.16 показано, что в группе переключателей могут быть элементы, и не являющиеся RadioButton. Более того, следует знать, что группа переключателей обеспечивает выбор единственного варианта только из тех вариантов переключателя, которые находятся в собственном контейнере этой группы. Это означает, что переключатель с ID anotherRadBtn не будет относиться к группе переключателей, показанных в листинге 4.16, так как не является дочерним элементом этой группы.

Необходимо также отметить, что RadioGroup можно управлять программными средствами. Например, можно получить ссылку на группу переключателей и добавить в нее еще один переключатель (или другой элемент управления):

```
RadioGroup rdgrp = (RadioGroup)findViewById(R.id.rdGrp);
RadioButton newRadioBtn = new RadioButton(this);
newRadioBtn.setText("Pork");
rdgrp.addView(newRadioBtn);
```

В итоге, когда пользователь выберет одно из положений переключателя, выбор нельзя будет отменить. Единственный способ, позволяющий программными средствами отменить выбор, — это применить к переключателю метод clearCheck(), позволяющий сбросить любой сделанный выбор.

Элементы управления списки

В Android SDK содержится несколько видов списков. На рис. 4.6 показан элемент управления ListView, который мы обсудим в данном разделе.

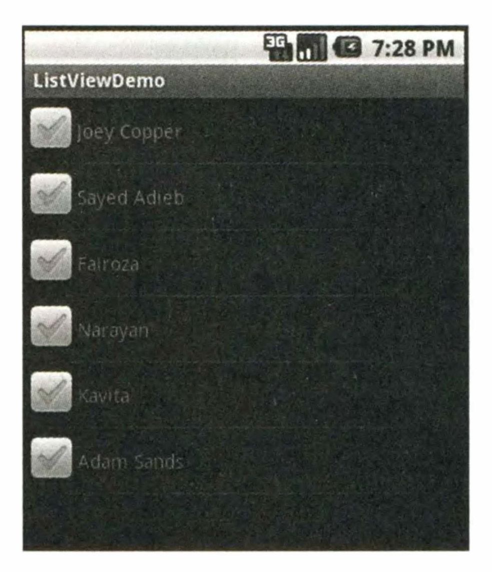


Рис. 4.6. Использование элемента управления ListView

ListView позволяет отобразить вертикальный список элементов. Обычно ListView используется при написании нового явления, дополняющего android.app.ListActivity.

ListActivity содержит ListView, и вы определяете данные, которые будут входить в ListView, вызывая метод setListAdapter(). В данном примере мы заполним весь экран ListView, так что не потребуется указывать ListView в основном XML-файле шаблона. Но нам будет нужен XML-шаблон для каждой строки. В листинге 4.17 показан файл шаблона строки, а также код Java для ListActivity.

Листинг 4.17. Добавление элементов в ListView

```
<?xml version="1.0" encoding="utf-8"?>
<!--Этот файл находится в /res/layout/list_item.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">

<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/row_chbox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
/>

<TextView android:id="@+id/row_tv" android:layout_width="wrap_content"
   android:layout_height="wrap_content"
/>
</LinearLayout>
public class ListDemoActivity extends ListActivity
{
   private SimpleCursorAdapter adapter;

   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      Cursor c = getContentResolver().query(People.CONTENT_URI,
                null, null, null, null);
      startManagingCursor(c);
      String[] cols = new String[]{People.NAME};
      int[] names = new int[]{R.id.row_tv};
      adapter = new SimpleCursorAdapter
                (this,R.layout.list_item,c,cols,names);
      this.setListAdapter(adapter);
   }
}
```

В листинге 4.17 создается элемент управления ListView, в который заносится список контактов, сохраненных в устройстве. Слева от каждого контакта находится элемент управления флажок. Как было указано выше, мы строим работу следующим образом: дополняем ListActivity, а затем устанавливаем адаптер списка. Это делается путем вызова метода setListAdapter() для явления. Например, мы запрашиваем у устройства список контактов, а потом создаем проекцию, при помощи которой выбираем только имена контактов, — в проекции задаются интере-

сующие нас столбцы. Затем мы сопоставляем имена с данными элемента управления TextView. Далее мы создаем адаптер курсора и устанавливаем адаптер списка. Класс адаптера достаточно интеллектуален, чтобы брать строки из источника данных и извлекать имя каждого контакта, а затем заполнять полученными такими образом именами пользовательский интерфейс.

Для выполнения поставленной задачи нам нужно сделать еще одну вещь. Поскольку в этом примере мы демонстрируем доступ к базе данных, в которой содержатся контакты данного мобильного телефона, необходимо запросить разрешение на это. Тема, касающаяся безопасности, более подробно рассмотрена в главе 7, но пока мы просто покажем, как создать ListView. Дважды щелкните на файле AndroidManifest.xml этого проекта, затем откройте вкладку **Permissions** (Права доступа). Нажмите кнопку **Add** (Добавить), выберите **Uses Permissions** (Использовать права доступа), а затем нажмите **OK**. Пролистайте список **Name** (Имя), пока не найдете **android.permission.READ_CONTACTS**. Окно Eclipse должно выглядеть как на рис. 4.7. Затем сохраните файл AndroidManifest.xml. Теперь это приложение можно запустить в эмуляторе. Возможно, вам потребуется добавить несколько контактов при помощи приложения Contacts, прежде чем все имена отобразятся в этом приложении.

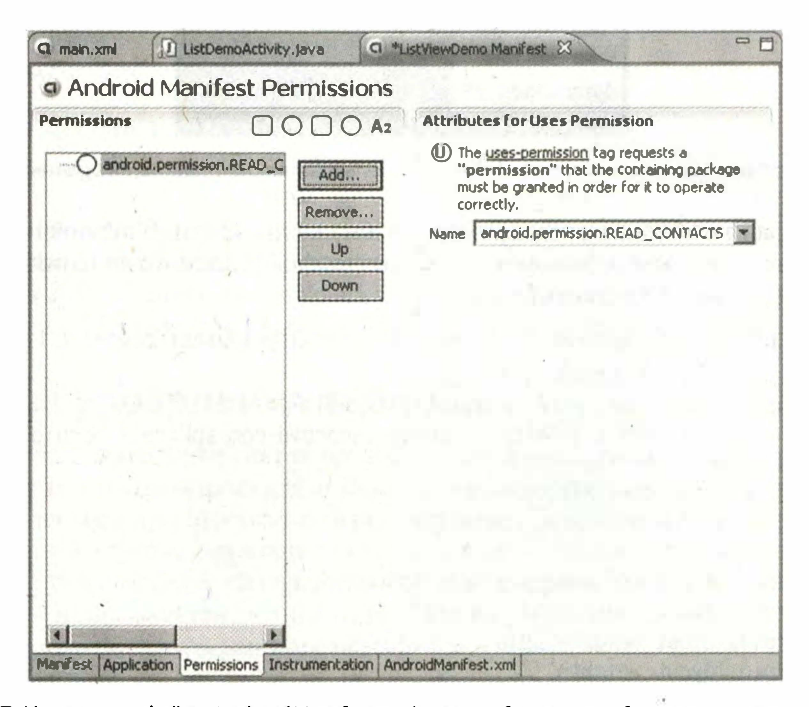


Рис. 4.7. Изменение файла AndroidManifest.xml таким образом, чтобы приложение работало

Как видите, метод onCreate() не задает внешний вид содержимого явления. Напротив, так как базовый класс ListActivity уже содержит ListView, нужно просто предоставить данные для ListView. Если вы хотите добавить в шаблон дополнительные элементы управления, можете взять XML-файл шаблона, вставить в него ListView и добавить необходимые элементы управления.

Например, можно добавить в пользовательский интерфейс ниже ListView кнопку, нажатие которой будет подтверждать выполнение операции над выделенными элементами, как это показано на рис. 4.8.

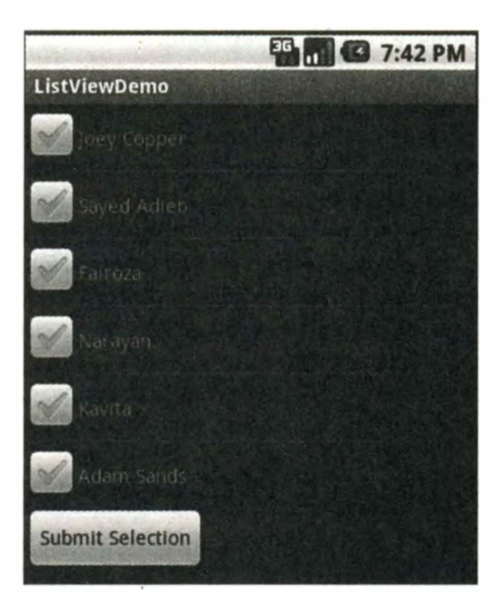


Рис. 4.8. Дополнительная кнопка, позволяющая пользователю отправить выбранные элементы

XML-файл шаблона в этом примере разбит на две части. В первой содержится определение объектов пользовательского интерфейса данного явления — ListView и кнопки (см. рис. 4.8 и листинг 4.18).

Листинг 4.18. Переопределение ListView, на который указывает ссылка с ListActivity

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Этот файл расположен в /res/layout/list.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">

   <ListView android:id="@android:id/list"
      android:layout_width="fill_parent"
      android:layout_height="0dip"
      android:layout_weight="1"
      android:stackFromBottom="true"
      android:transcriptMode="normal"/>

   <Button android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Submit Selection" />

</LinearLayout>
```

Во втором файле содержится определение элементов списка, аналогичное определению, данному в листинге 4.17. В таком случае явление будет выглядеть как в листинге 4.19.

Листинг 4.19. Определение вида содержимого ListActivity

```
public class ListDemoActivity extends ListActivity
{
   private SimpleCursorAdapter adapter;
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.list);

      Cursor c = getContentResolver().query(People.CONTENT_URI,
               null, null, null, null);
      startManagingCursor(c);

      String[] cols = new String[]{People.NAME};
      int[] names = new int[]{R.id.row_tv};
      adapter = new SimpleCursorAdapter
               (this,R.layout.list_item,c,cols,names);
      this.setListAdapter(adapter);
   }
}
```

В листинге 4.19 показано, что класс явления вызывает setContentView(), чтобы установить вид пользовательского интерфейса для данного явления. Здесь же при создании адаптера определяется файл шаблона для элементов списка (подробнее мы поговорим об адаптерах в разделе «Адаптеры» ближе к концу этой главы).

Элементы управления таблицы

В большинстве наборов виджетов предоставляется один или несколько элементов управления, предназначенных для показа таблиц. В Android есть элемент управления GridView, позволяющий отображать данные в сетке. Обратите внимание — под термином «данные» в этом контексте могут пониматься текст, изображения и т. д.

Элемент управления GridView отображает информацию в форме таблицы. Обычно GridView используется, чтобы задать сетку таблицы в XML-шаблоне (листинг 4.20), а затем связать данные с таблицей при помощи android.widget.ListAdapter. Чтобы этот пример работал, не забудьте добавить в файл AndroidManifest.xml тег, регламентирующий работу этого примера.

Листинг 4.20. Определение GridView в XML-шаблоне и связанном с ним коде Java

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Этот файл расположен в /res/layout/gridview.xml -->
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/dataGrid"
   android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:padding="10px"
android:verticalSpacing="10px"
android:horizontalSpacing="10px"
android:numColumns="auto_fit"
android:columnWidth="100px"
android:stretchMode="columnWidth"
android:gravity="center"
/>

protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.gridview);
      GridView gv = (GridView)this.findViewById(R.id.dataGrid);

      Cursor c = getContentResolver().query(People.CONTENT_URI,
            null, null, null, null);
      startManagingCursor(c);

      String[] cols = new String[]{People.NAME};
      int[] names = new int[]{android.R.id.text1};

      SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
         android.R.layout.simple_list_item_1 ,c,cols,names);

      gv.setAdapter(adapter);

}
```

В листинге 4.20 в XML-шаблоне определяется простой элемент GridView. Затем таблица загружается в качестве видимого содержимого явления. Полученный таким образом пользовательский интерфейс показан на рис. 4.9.

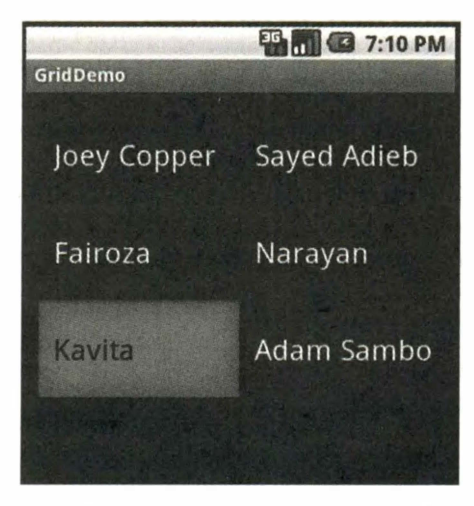


Рис. 4.9. Элемент GridView, заполненный контактной информацией

В таблице, показанной на рис. 4.9, отображаются названия контактов, содержащихся в памяти устройства. Ранее мы решили отобразить названия контактов в TextView, но можем создать и таблицу, которая будет заполнена изображениями и другим контентом. Следует отметить, что в этом примере использован другой, упрощенный метод. Чтобы не создавать специальный файл шаблона для элементов таблицы, мы воспользуемся одним из заданных шаблонов, предлагаемых в Android. Обратите внимание на префикс ресурсов, используемых при создании шаблонов элементов таблицы и полей таблицы — android:. Android ищет такие ресурсы не в нашем локальном каталоге /res, а в собственном. Вы можете просмотреть эту папку, перейдя в каталог **Android SDK** и открыв platforms/<android-version>/data/res/layout. Здесь вы найдете файл simple_list_item_1.xml, в котором будет задан простой TextView, чей android:id имеет значение @android:id/text1. Именно поэтому мы указали android.R.id.text1 для ID имен адаптера курсора.

Характерная деталь, связанная с GridView, заключается в том, что таблица использует адаптер ListAdapter. Списки обычно одномерные, а таблицы — двухмерные. На основании этого можно сделать вывод, что в таблице на самом деле отображаются списковые данные. Если вызвать метод getSelection(), система вернет натуральное число, соответствующее номеру выбранного элемента. Подобным образом, чтобы произвести выбор из таблицы, можно вызвать setSelection() с номером элемента, который вы хотите выбрать.

Элементы управления датой и временем

Во многих наборах виджетов содержатся инструменты для управления датой и временем. В Android также предлагаются такие элементы управления, некоторые из них будут рассмотрены в этом разделе. В частности, мы поговорим об элементах управления DatePicker, TimePicker, AnalogClock и DigitalClock.

Элементы управления DatePicker и TimePicker

Из названия понятно, что элемент управления DatePicker используется для выбора даты, а TimePicker — для выбора времени. В листинге 4.21 и на рис. 4.10 показаны примеры таких элементов управления.

Листинг 4.21. Элементы управления DatePicker и TimePicker в XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent">

 <DatePicker android:id="@+id/datePicker"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />

 <TimePicker android:id="@+id/timePicker"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content" />

</LinearLayout>
```

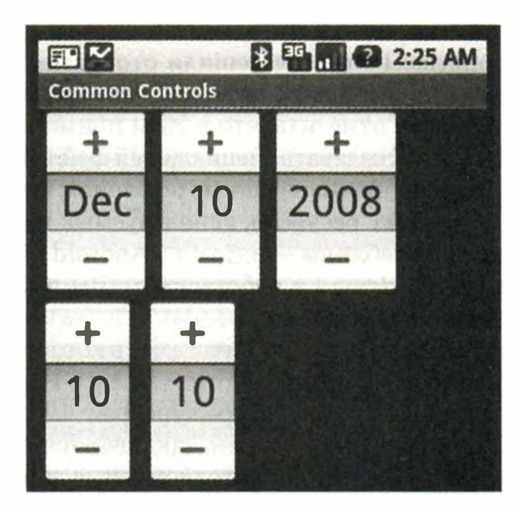


Рис. 4.10. Пользовательские интерфейсы DatePicker и TimePicker

Если внимательно рассмотреть шаблон XML, становится ясно, что определять такие элементы управления довольно просто. Однако пользовательский интерфейс выглядит немного непропорциональным. Оба элемента управления кажутся слишком крупными, но, поскольку различные мобильные устройства значительно отличаются друг от друга, мы не можем с определенностью судить, как именно эти элементы управления будет воспринимать пользователь.

Интерфейсами `DatePicker` и `TimePicker` можно управлять с помощью методов программирования, как и другими элементами, входящими в инструментарий Android. В частности, средства программирования можно использовать для инициализации элементов или получения от них данных. Например, в листинге 4.22 показано, как инициализировать такие элементы управления.

Листинг 4.22. Инициализация DatePicker и TimePicker соответственно со значениями даты и времени

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.datetime);

    DatePicker dp = (DatePicker)this.findViewById(R.id.datePicker);
    dp.init(2008, 11, 10, null);

    TimePicker tp = (TimePicker)this.findViewById(R.id.timePicker);
    tp.setIs24HourView(true);
    tp.setCurrentHour(new Integer(10));
    tp.setCurrentMinute(new Integer(10));
}
```

В листинге 4.22 для `DatePicker` установлена дата 10 ноября 2008 года. Значение времени — 10 часов 10 минут. Обратите внимание — в элементе управления поддерживается 24-часовая временная шкала. Если не установить значения для этих

элементов управления, значениями по умолчанию будут актуальные дата и время, зафиксированные в устройстве.

Наконец, необходимо отметить, что в Android эти элементы управления также могут быть представлены как модальные окна — `DatePickerDialog` и `TimePickerDialog`. Элементы управления в такой форме удобны в тех случаях, когда вы хотите привлечь к элементу внимание пользователя и заставить его сделать выбор. Более подробно диалоговые окна будут рассмотрены в главе 5.

Элементы управления AnalogClock и DigitalClock

В Android также имеются элементы управления `AnalogClock` и `DigitalClock` (рис. 4.11).



Рис. 4.11. Использование AnalogClock и DigitalClock

На рисунке видно, что аналоговые часы, используемые в Android, — это часы с двумя стрелками, часовой и минутной. На цифровых часах, кроме часов и минут, также отсчитываются секунды.

Два указанных элемента управления не слишком интересны, так как на них нельзя изменять дату и время. Иными словами, это просто часы, которые могут всего лишь отсчитывать время. Это означает, что если вам понадобится изменить дату и время, то лучше остановиться на `DatePicker`/`TimePicker` или `DatePickerDialog`/`TimePickerDialog`.

Другие интересные элементы управления, имеющиеся в Android

Элементы управления, изученные нами выше, являются основополагающими компонентами любого приложения Android. Кроме них, в Android применяются и другие интересные элементы управления. В этом разделе мы кратко их рассмотрим.

Элемент управления MapView

Элемент управления `com.google.android.maps.MapView` может отображать карту. Этот элемент управления можно инстанцировать либо через шаблон XML, либо через

код, но то явление, которое будет использовать этот элемент, должно быть добавлено к MapActivity. Последний отвечает за обработку многопоточных запросов и выполняет такие задачи, как загрузка карты, осуществление кэширования и т. д.

В листинге 4.23 показан пример инстанцирования MapView.

Листинг 4.23. Создание элемента управления MapView в шаблоне XML

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <com.google.android.maps.MapView
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:enabled="true"
      android:clickable="true"
      android:apiKey="myAPIKey"
      />

</LinearLayout>
```

Элемент управления MapView будет подробно рассмотрен в главе 7, когда мы будем говорить о службах, зависящих от местоположения. Там же мы изучим, как получить собственный ключ к программному интерфейсу для построения карт (mapping API key).

Элемент управления галерея

Элемент Gallery — это горизонтальный прокручиваемый список, позволяющий поместить в фокус тот элемент списка, который находится в центре. Обычно этот элемент управления работает как фотогалерея в сенсорном режиме. Галерею можно инстанцировать либо через шаблон XML, либо через код:

```
<Gallery
   android:id="@+id/galleryCtrl"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
/>
```

Работа с галереей похожа на работу со списком. Вы получаете ссылку на галерею, затем вызываете метод setAdapter() для занесения данных, потом регистрируете события, которые должны происходить при выборе элемента.

Элемент управления счетчик

Элемент управления счетчик (spinner) можно инстанцировать либо через шаблон XML, либо через код:

```
<Spinner
   android:id="@+id/spinner"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

Работа со счетчиками также похожа на работу со списками. То есть вы получаете ссылку на счетчик, затем вызываете метод `setAdapter()` для занесения данных, потом регистрируете события, которые должны происходить при выборе элемента. Мы используем счетчик в одном из примеров далее в этой главе, в разделе «Знакомство с ArrayAdapter».

На этом мы завершаем обсуждение элементов управления, используемых в Android. Как было указано в начале этой главы, для создания пользовательских интерфейсов в Android необходимо научиться работать с двумя сущностями: элементами управления и диспетчерами шаблонов.

Диспетчеры шаблонов

Как и в Swing, в Android используются классы представления (view classes), которые действуют как контейнеры для видов. Такие контейнеры видов называются шаблонами (layouts), или диспетчерами шаблонов (layout managers), и каждый из них реализует специфический метод управления размером и расположением дочерних элементов. Например, класс `LinearLayout` позволяет располагать дочерние элементы горизонтально или вертикально, один за другим.

В табл. 4.2 перечислены диспетчеры шаблонов, поставляемые вместе с Android SDK.

Таблица 4.2. Диспетчеры шаблонов, используемые в Android

Диспетчер шаблонов	Описание
LinearLayout	Располагает дочерние элементы по горизонтали или по вертикали
TableLayout	Располагает дочерние элементы в форме таблицы
RelativeLayout	Располагает дочерние элементы относительно друг друга или относительно родительского элемента
FrameLayout	Позволяет динамически изменять элемент(ы) управления в шаблоне

В следующих разделах мы подробно опишем эти диспетчеры шаблонов. Ранее также использовался еще один диспетчер шаблонов — `AbsoluteLayout`, но он уже устарел и в нашей книге рассматриваться не будет.

Диспетчер шаблонов LinearLayout

Диспетчер шаблонов `LinearLayout` используется чаще всего. Он позволяет расположить находящиеся в нем дочерние элементы по горизонтали или по вертикали в зависимости от значения свойства `orientation`. В листинге 4.24 показан диспетчер `LinearLayout` с горизонтальной конфигурацией.

Листинг 4.24. Диспетчер шаблонов LinearLayout с горизонтальной конфигурацией

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">

<!-- сюда добавляются дочерние элементы-->

</LinearLayout>
```

Можно создать LinearLayout и с вертикальной ориентацией — для этого в качестве значения orientation устанавливается vertical.

Важность (weight) и выравнивание (gravity)

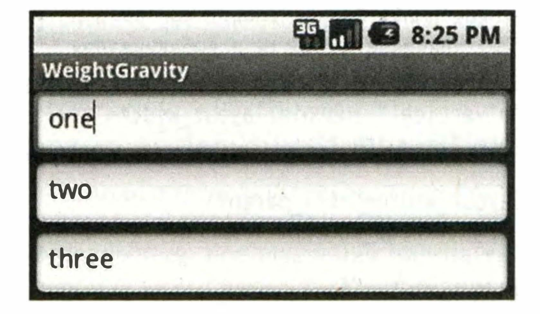
Атрибут orientation — это первый важный атрибут, распознаваемый менеджером шаблонов LinearLayout. Другими важными свойствами, которые могут влиять на размер и положение дочерних элементов, являются *важность* (weight) и *выравнивание* (gravity). Weight используется для присвоения элементу показателя важности, отличающего его от других элементов, находящихся в контейнере. Предположим, в контейнере находится три элемента управления: первый имеет важность 1 (максимальное возможное значение), а два других имеют значение 0. В этом случае элемент управления, который имеет значение важности 1, займет в контейнере все свободное пространство. Выравнивание (gravity) — это ориентация в контейнере (alignment). Например, вы хотите выровнять текст надписи по правому краю, тогда свойство gravity будет иметь значение right. Набор значений для gravity достаточно ограничен. В него входят left, center, right, top, bottom, center_vertical, clip_horizontal и еще некоторые. Эти и другие значения выравнивания подробно описаны на специальных справочных сайтах.

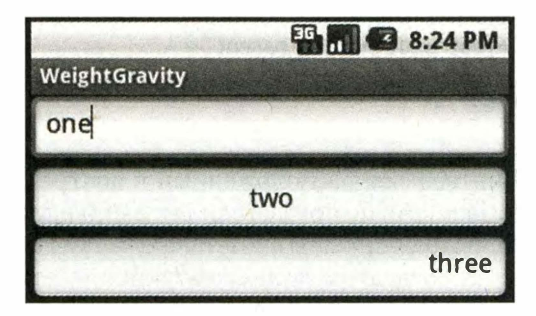
ПРИМЕЧАНИЕ

Диспетчеры шаблонов дополняют android.widget.ViewGroup, подобно многим классам контейнеров, предназначенных для управления, например ListView. Но хотя диспетчеры шаблонов и классы для элементов управления и дополняют один и тот же класс, классы диспетчеров шаблонов работают только с размерами и расположением элементов управления и никак не связаны с взаимодействием пользователя с дочерними элементами. Например, сравните LinearLayout с элементом управления ListView. На экране они выглядят схоже, так как и тот и другой располагают находящиеся в них элементы по вертикали. Но элемент управления ListView предоставляет пользователю интерфейс API, при помощи которого можно делать выбор, а LinearLayout — нет. Иными словами, контейнер, предназначенный для управления (ListView) поддерживает работу пользователя с элементами, находящимися в этом контейнере, а диспетчер шаблонов (LinearLayout) работает только с размерами и расположением элементов.

Теперь рассмотрим, как свойства важности и выравнивания используются на практике (рис. 4.12).

На рис. 4.12 показаны три пользовательских интерфейса, в которых используется LinearLayout с различными настройками важности и выравнивания. В левом интерфейсе применяются стандартные настройки важности и выравнивания. XML-шаблон первого пользовательского интерфейса показан в листинге 4.25.





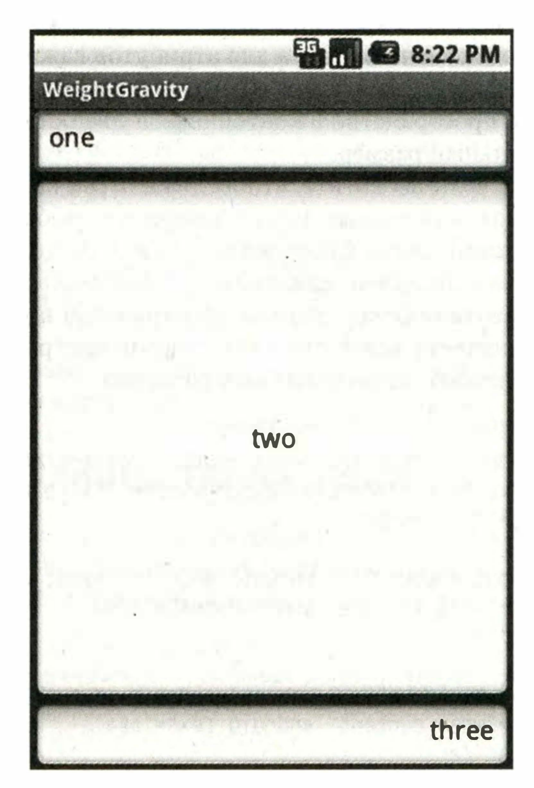


Рис. 4.12. Использование менеджера шаблонов LinearLayout

Листинг 4.25. Три текстовых поля, расположенных вертикально в LinearLayout, с применением стандартных значений важности и выравнивания

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <EditText android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="one"/>
   <EditText android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="two"/>
   <EditText android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="three"/>
</LinearLayout>
```

Пользовательский интерфейс, расположенный в центре рис. 4.12, использует стандартное значение важности, но показатели выравнивания android:gravity для элементов управления, находящихся в контейнере, имеют значения left, center и right соответственно. В последнем примере атрибут android:layout_weight центрального компонента имеет значение 1.0, а для остальных компонентов значение оставлено без изменений — 0.0 (листинг 4.26). Если присвоить атрибуту важности центрального компонента значение 1.0, а для атрибутов важности остальных элементов оставить стандартные значения (0.0), центральный компонент займет все оставшееся свободное пространство в контейнере, а два оставшихся компонента должны иметь оптимальный размер.

Подобным образом, если вы хотите, чтобы два из трех компонентов, расположенных в контейнере, заняли равные доли оставшегося свободного пространства, важность двух этих компонентов будет равна 1.0, а оставшийся третий элемент будет иметь стандартное значение важности — 0.0. Наконец, если необходимо, чтобы все три компонента поровну делили пространство в контейнере, каждый элемент будет иметь значение важности 1.0. С такими настройками все элементы в контейнере будут занимать одинаковое пространство.

Листинг 4.26. LinearLayout с настройками важности

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical" android:layout_width="fill_parent"
 android:layout_height="fill_parent">

 <EditText android:layout_width="fill_parent" android:layout_weight="0.0"
 android:layout_height="wrap_content" android:text="один"
 android:gravity="left"/>

 <EditText android:layout_width="fill_parent" android:layout_weight="1.0"
 android:layout_height="wrap_content" android:text="два"
 android:gravity="center"/>

 <EditText android:layout_width="fill_parent" android:layout_weight="0.0"
```

```
 android:layout_height="wrap_content" android:text="три"
 android:gravity="right"
 />
</LinearLayout>
```

android:gravity против android:layout_gravity

Обратите внимание: в Android определяются два сходных атрибута выравнивания: android:gravity и android:layout_gravity. Разница заключается в том, что android:gravity — это настройка, используемая видом, а android:layout_gravity применяется контейнером (android.view.ViewGroup). Например, можно установить для android:gravity значение center, чтобы текст в EditText был выровнен в элементе управления по центру. Аналогичным образом можно выровнять EditText по правому краю LinearLayout (который является контейнером), установив android:layout_gravity="right" (рис. 4.13 и листинг 4.27).



Рис. 4.13. Применение настроек выравнивания

Листинг 4.27. Объяснение разницы между android:gravity и android:layout_gravity

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <EditText android:layout_width="wrap_content" android:gravity="center"
   android:layout_height="wrap_content" android:text="один"
   android:layout_gravity="right"/>
</LinearLayout>
```

На рис. 4.13 показано, что текст выровнен в EditText по центру, а сам EditText выровнен по правому краю LinearLayout.

Диспетчер шаблонов TableLayout

Диспетчер шаблонов TableLayout дополняет LinearLayout. Диспетчер шаблонов TableLayout располагает свои дочерние элементы в строках и столбцах. В листинге 4.28 показан соответствующий пример.

Листинг 4.28. Простой шаблон TableLayout

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

   <TableRow>
```

```
        <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="First Name:"/>

        <EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Barack"/>
    </TableRow>

    <TableRow>
        <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Last Name:"/>

        <EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Obama"/>
    </TableRow>

</TableLayout>
```

Для работы с TableLayout создается экземпляр TableLayout, в котором затем располагаются элементы TableRow. В TableRow содержатся элементы управления из данной таблицы. Пользовательский интерфейс, соответствующий листингу 4.28, показан на рис. 4.14.



Рис. 4.14. Диспетчер шаблонов TableLayout

Поскольку содержимое TableLayout определяется взаимным расположением строк и столбцов, Android узнает количество столбцов в таблице, находя строку с максимальным количеством ячеек. Например, в листинге 4.29 создается таблица с двумя строками, где в одной строке находится две ячейки, а в другой — три (рис. 4.15). В таком случае Android создает таблицу с двумя строками и тремя столбцами. Ячейка в последнем столбце первой строки будет пуста.

Листинг 4.29. Определение нерегулярной таблицы

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent">

  <TableRow>
     <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="First Name:"/>

     <EditText android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Barack"/>
```

```
 </TableRow>

 <TableRow>
     <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Last Name:"/>

     <EditText android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Hussein"/>

     <EditText android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Obama"/>
 </TableRow>

</TableLayout>
```



Рис. 4.15. Шаблон TableLayout для нерегулярной таблицы

В листингах 4.28 и 4.29 мы заполнили `TableLayout` элементами `TableRow`. Хотя обычно принято действовать именно так, вы можете указать любой `android.widget.View` в качестве дочернего элемента таблицы. Например, в листинге 4.30 создается таблица, первая строка которой — `EditText` (рис. 4.16).

Листинг 4.30. Использование EditText вместо TableRow

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 android:stretchColumns="0,1,2">

<EditText
 android:text="Full Name:"/>

 <TableRow>
     <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Barack"/>

 <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Hussein"/>

     <TextView android:layout_width="wrap_content"
     android:layout_height="wrap_content" android:text="Obama"/>
 </TableRow>

</TableLayout>
```



Рис. 4.16. EditText в качестве дочернего элемента TableLayout

Пользовательский интерфейс, описанный в листинге 4.30, показан на рис. 4.16. Обратите внимание: EditText распространяется на всю ширину экрана, хотя это и не указано в XML-шаблоне. Такое явление объясняется тем, что дочерние элементы TableLayout всегда занимают целую строку. Иными словами, для дочерних элементов TableLayout нельзя задать android:layout_width="wrap_content" — они обязательно принимают fill_parent. Однако они могут принимать android:layout_height.

Поскольку во время разработки таблицы не всегда известно, какое содержимое она будет иметь, в TableLayout предусмотрено несколько атрибутов, которые помогают управлять расположением элементов в таблице. Например, в листинге 4.30 в TableLayout устанавливается свойство android:stretchColumns со значением "0,1,2". Таким образом, система подсказывает TableLayout, что при необходимости столбцы 0, 1 и 2 можно расширить в зависимости от содержимого таблицы.

Аналогичным образом вы можете установить свойство android:shrinkColumns, позволяющее заверстывать на новую строку содержимое одного или нескольких столбцов, если для других столбцов требуется больше места. Еще можно установить android:collapseColumns, чтобы сделать столбцы невидимыми. Обратите внимание: идентификация столбцов происходит по схеме индексации с основанием 0.

В TableLayout предусмотрено также свойство android:layout_span. Оно используется для того, чтобы одна ячейка могла расширяться на несколько столбцов. Это поле похоже на свойство colspan, используемое в HTML.

Иногда может также понадобиться сделать отступы между фрагментами текста, находящимися в ячейке или в элементе управления. В Android SDK такая функция поддерживается при помощи android:padding и соседних смежных свойств. Android:padding позволяет регулировать ширину пространства между внешними границами вида и его содержимым (листинг 4.31).

Листинг 4.31. Работа с android:padding

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <EditText android:layout_width="wrap_content"
   android:layout_height="wrap_content" android:text="one"
   android:padding="40px" />
</LinearLayout>
```

В листинге 4.31 для отступа устанавливается значение 40px. Таким образом, создается пустое пространство шириной 40 пикселов между внешней границей элемента управления EditText и текстом, находящимся внутри этого элемента. На рис. 4.17 показаны два варианта одного и того же элемента EditText с разными значениями отступа. Пользовательский интерфейс, приведенный слева, не имеет свойства padding, а в правом интерфейсе действует настройка android:padding="40px".



Рис. 4.17. Использование отступа

Android:padding устанавливает отступ со всех четырех сторон: слева, справа, сверху и снизу. Чтобы определять отступ для каждой стороны отдельно, используйте android:leftPadding, android:rightPadding, android:topPadding и android:bottomPadding.

В Android также определяется android:layout_margin, свойство, похожее на android:padding. На самом деле android:padding/android:layout_margin аналогично android:gravity/android:layout_gravity. Но первые два свойства используются с видом, а вторые — с контейнером.

Наконец, значение отступа всегда устанавливается в единицах измерения (dimension type). В Android поддерживаются следующие единицы измерения.

- *Пикселы* — сокращенное обозначение px. Данный тип размерности соответствует пикселам, фактически отображаемым на экране.
- *Дюймы* — сокращенное обозначение in.
- *Миллиметры* — сокращенное обозначение mm.
- *Точки* — сокращенное обозначение pt. Точка равна 1/72 дюйма.
- *Пикселы, не зависящие от плотности экрана,* — сокращенное обозначение dip или dp. При таком типе размерности плотность экрана 160 точек используется для создания базовой системы координат, а затем сопоставляется с имеющимся экраном. Например, если применяется экран шириной 160 пикселов, соотношение между dip и px составит 1:1.
- *Пикселы, не зависящие от масштаба,* — сокращенное обозначение sp. Обычно используется с типами шрифта. При таком типе размерности система учитывает пользовательские настройки и размер шрифта для определения точного размера.

Обратите внимание — описанные выше единицы измерения не являются специфичными для отступа — любое поле Android, принимающее размерную величину

(например, android:layout_width или android:layout_height), поддерживает и такие единицы измерения.

Диспетчер шаблонов RelativeLayout

Еще один интересный диспетчер шаблонов — это RelativeLayout. Из его названия понятно, что в данном диспетчере шаблонов применяется политика, при которой элементы управления располагаются в контейнере либо относительно самого контейнера, либо относительно другого элемента управления в контейнере. В листинге 4.32 и на рис. 4.18 показан соответствующий пример.

Листинг 4.32. Использование диспетчера шаблонов RelativeLayout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content">

<TextView android:id="@+id/userNameLbl"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Username: "
 android:layout_alignParentTop="true" />

<EditText android:id="@+id/userNameText"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_below="@id/userNameLbl" />

<TextView android:id="@+id/pwdLbl"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_below="@id/userNameText"
 android:text="Password: " />

<EditText android:id="@+id/pwdText"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_below="@id/pwdLbl" />

<TextView android:id="@+id/pwdHintLbl"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_below="@id/pwdText"
 android:text="Password Criteria... " />

<TextView android:id="@+id/disclaimerLbl"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentBottom="true"
 android:text="Use at your own risk... " />

</RelativeLayout>
```

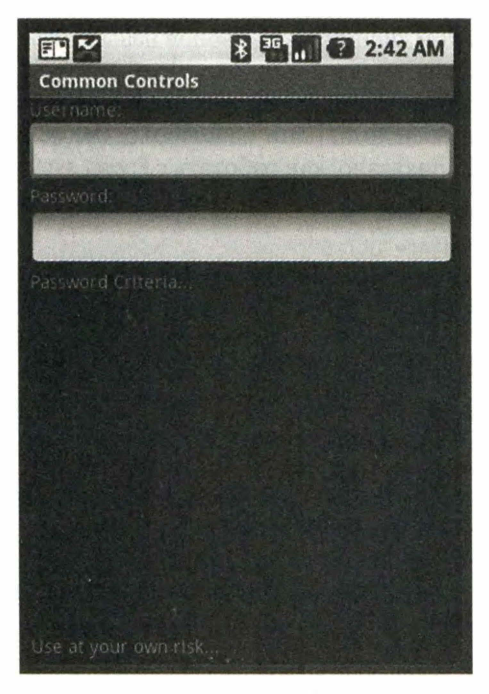


Рис. 4.18. Пользовательский интерфейс, скомпонованный при помощи диспетчера шаблонов RelativeLayout

Пользовательский интерфейс, показанный в предыдущем примере, выглядит как простая форма для входа в систему (login form). Обозначение поля для имени пользователя расположено на самом верху контейнера, так как мы установили для `android:layout_alignParentTop` значение `true`. В свою очередь, строка для ввода имени пользователя расположена под обозначением этого поля, поскольку мы задали `android:layout_below`. Обозначение для поля с паролем находится еще ниже, а стандартная фраза об отказе от ответственности располагается в самом низу контейнера, поскольку мы задали для `android:layout_alignParentBottom` значение `true`.

Кроме этих трех атрибутов расположения, вы можете указать `layout_above`, `layout_toRightOf`, `layout_toLeftOf`, `layout_centerInParent` и еще некоторые атрибуты. Работать с `RelativeLayout` просто, так как этот диспетчер шаблонов очень прост. Как только вы начнете им пользоваться, он станет вашим любимым диспетчером шаблонов — вы будете прибегать к нему снова и снова.

Диспетчер шаблонов FrameLayout

Диспетчеры шаблонов, рассмотренные нами выше, используют различные стратегии компоновки элементов. Иными словами, в каждом диспетчере шаблонов предусмотрен особый механизм, в соответствии с которым дочерние элементы шаблона располагаются и выравниваются на экране. Используя такие диспетчеры шаблонов, вы можете разместить на экране несколько элементов управления одновременно, и каждый элемент будет занимать определенный фрагмент экрана. В Android есть и такой диспетчер шаблонов, который позволяет многократно отображать один и тот же

конкретный элемент. Такой диспетчер шаблонов называется FrameLayout. Этот вспомогательный класс, относящийся к построению шаблонов, обычно применяется для динамического отображения отдельно взятого вида, но вы можете заполнить такой шаблон несколькими элементами, одни из которых будут видимы, а другие — невидимы. В листинге 4.33 показано, как работать с FrameLayout.

Листинг 4.33. Заполнение FrameLayout

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/frmLayout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

        <ImageView
            android:id="@+id/oneImgView" android:src="@drawable/one"
            android:scaleType="fitCenter"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"/>
        <ImageView
            android:id="@+id/twoImgView" android:src="@drawable/two"
            android:scaleType="fitCenter"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:visibility="gone" />

</FrameLayout>

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.frame);

    ImageView one = (ImageView)this.findViewById(R.id.oneImgView);
    ImageView two = (ImageView)this.findViewById(R.id.twoImgView);

    one.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View view) {
            ImageView two = (ImageView)FramelayoutActivity.this.
                findViewById(R.id.twoImgView);

            two.setVisibility(View.VISIBLE);

            view.setVisibility(View.GONE);
        }});

    two.setOnClickListener(new OnClickListener(){

        @Override
```

```
        public void onClick(View view) {
            ImageView one = (ImageView)FramelayoutActivity.
                this.findViewById(R.id.oneImgView);

            one.setVisibility(View.VISIBLE);

            view.setVisibility(View.GONE);
        }});
}
```

В листинге 4.33 показан файл шаблона, а также метод onCreate(), используемый с явлением. Идея примера заключается в том, чтобы загрузить во FrameLayout два объекта ImageView, причем в каждый момент времени должен быть виден только один объект ImageView. Если при работе с интерфейсом пользователь щелкает на видимом ImageView, мы скрываем этот объект, а вместо него отображаем тот ImageView, который до этого был скрыт.

Теперь рассмотрим листинг 4.33 более подробно и начнем с шаблона. Как видите, мы задаем FrameLayout с двумя объектами ImageView (ImageView — это элемент управления, предназначенный для показа изображений). Обратите внимание — свойство visibility второго объекта ImageView имеет значение gone, благодаря чему данный элемент управления и является невидимым. Теперь изучим метод onCreate(). В методе onCreate() регистрируются слушатели, которые будут реагировать на события нажатий, фиксируемые на объектах ImageView. Обработчик щелчков (click handler) позволяет скрыть один объект ImageView и показать другой.

Как было сказано выше, обычно FrameLayout используется в тех случаях, когда требуется динамически создать из контента отдельно взятого вида элемент управления. Обычно такой элемент управления имеет много дочерних элементов, как мы могли убедиться на примере, показанном выше. В листинге 4.33 в шаблон добавляются два элемента управления, но в каждый момент виден только один из них. Правда, в FrameLayout не обязательно требуется отображать только один элемент управления в конкретное время. Если добавить в шаблон много элементов управления, FrameLayout просто сложит их в стек (можно сказать, в стопку) один поверх другого, причем выше всех будет находиться тот элемент управления, который был добавлен последним. Так можно создать очень интересный пользовательский интерфейс. Например, на рис. 4.19 показан FrameLayout, в котором видимы два объекта ImageView. Как видите, эти элементы лежат друг на друге, и верхнее изображение частично прикрывает нижнее.

Еще одна интересная черта FrameLayout заключается в том, что при добавлении в шаблон более одного элемента управления размер шаблона рассчитывается как размер самого крупного элемента, находящегося в контейнере. На рис. 4.19 верхнее изображение на самом деле гораздо меньше, чем изображение под ним, но поскольку размер шаблона рассчитывается как размер самого крупного элемента, находящегося в контейнере, верхнее изображение растягивается.

Обратите внимание и на то, что, если вы поместите внутри FrameLayout один или несколько элементов, невидимых в начале работы, можно будет использовать метод setConsiderGoneChildrenWhenMeasuring(). Поскольку размер контейнера определяется по самому крупному дочернему элементу, вам будет сложно начинать работу, если

в начале процесса самый крупный элемент окажется невидимым. Поэтому, когда элемент станет видим, на экране отобразится только его часть. Чтобы гарантировать, что все элементы будут отображаться правильно, вызовите `setConsiderGoneChildrenWhenMeasuring()` и задайте ему значение `true`.

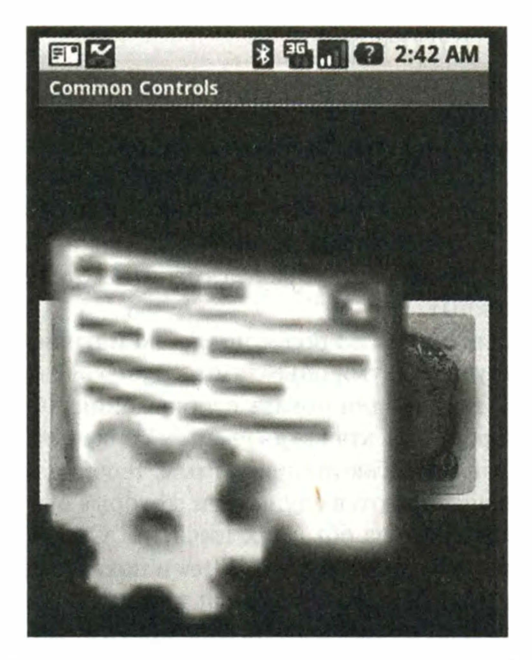


Рис. 4.19. FrameLayout, в котором находятся два объекта ImageView

Настройка расположения элементов для различных конфигураций устройств

Теперь вы знаете, что в Android содержатся диспетчеры шаблонов, при помощи которых вы можете строить пользовательские интерфейсы. Если вы на практике познакомились с теми диспетчерами шаблонов, которые мы обсудили выше, то знаете, что диспетчеры шаблонов можно комбинировать различными способами, чтобы получить интерфейс, который вам нужен. Но, даже имея в распоряжении полный набор диспетчеров шаблонов, может быть непросто построить пользовательский интерфейс — а потом еще и заставить его работать правильно. Это утверждение особенно справедливо, если вы программируете для мобильных устройств. Запросы со стороны пользователей и производителей мобильных устройств постоянно растут, и труд разработчика становится все более сложным.

Одна из проблем, возникающих при создании пользовательского интерфейса для мобильного устройства, заключается в том, что этот интерфейс будет отображаться на дисплеях с различными видами конфигурации. Например, как станет выглядеть пользовательский интерфейс, если приложение будет отображаться то

в вертикальном, то в горизонтальном формате? Если вы еще не сталкивались с такой ситуацией, то, вероятно, уже ломаете голову: как же справиться с настолько тривиальной проблемой? Но остается только радоваться, что в Android предоставляется поддержка и на этот случай.

Принцип действия таков: Android находит и загружает шаблоны из конкретных каталогов, в зависимости от конфигурации устройства. Существует три варианта конфигурации: вертикальный формат (portrait), горизонтальный формат (landscape) или квадратный формат (square). Чтобы можно было использовать различные варианты шаблонов при разных конфигурациях, вам потребуется создать отдельные каталоги для каждой конфигурации, из которых Android будет загружать подходящие шаблоны. Как вы уже знаете, по умолчанию шаблоны располагаются в каталоге `res/layout`. Для поддержки изображения в вертикальном формате (portrait display) создайте каталог `res/layout-port`. Для поддержки изображения в горизонтальном формате (landscape display) создайте каталог `res/layout-land`. А для квадратного формата создайте каталог `res/layout-square`.

Здесь может возникнуть резонный вопрос: «Если у меня есть три этих новых каталога для шаблонов, нужен ли мне стандартный каталог шаблонов `res/layout`?» Обычно нужен. Согласно логике разрешения ресурсов, действующей в Android, система сначала ищет ресурс в каталоге, специфичном для конкретной конфигурации. Если Android не находит там нужный ресурс, он переходит к стандартному каталогу шаблонов. Следовательно, в `res/layout` можно поместить определения шаблонов, задаваемые по умолчанию, а версии, которые вы настроили для различных конфигураций устройств, — в специально созданных для этого каталогах.

Обратите внимание: в Android SDK не предоставляется никаких программных интерфейсов, в которых можно было бы методами программирования указывать, какой вариант конфигурации следует загрузить. Система просто выбирает каталог в зависимости от конфигурации устройства. Однако ориентацию изображения можно определить и в коде, например, следующим образом:

```
import android.content.pm.ActivityInfo;
...
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
```

При наличии такого кода изображение обязательно будет выводиться на устройстве в горизонтальном формате. Можете попробовать использовать этот способ с проектами, созданными вами ранее. Добавьте этот код к методу `onCreate()` явления, запустите программу для просмотра изображения в режиме «вид сбоку».

Шаблон — это не единственный ресурс, определяемый конфигурацией. Существуют и иные квалификаторы конфигурации устройства, учитываемые при поиске ресурса, который будет использоваться. Все содержимое каталога `res` может иметь особенности, характерные для конкретной конфигурации. Например, чтобы при разных вариантах конфигурации загружались различные отрисовываемые объекты (drawables), создайте отдельные каталоги для `drawable-port`, `drawable-land` и `drawable-square`. Но и это еще не все. Полный список квалификаторов, которыми можно пользоваться при нахождении ресурсов, приведен в табл. 4.3.

Таблица 4.3. Квалификаторы ресурсов

Квалификатор	**Описание**
MCC и MNC	Мобильный код страны и код мобильной сети
Язык и регион	Двухбуквенный код языка, возможно, с добавлением 'r', и двухбуквенный код региона
Параметры экрана	Дает приближенное значение размера экрана, может принимать значения small, normal, large
Более широкие/более высокие экраны	Относится к соотношению сторон изображения, может принимать значения long, notlong
Ориентация экрана	Значения land, port, square
Плотность пикселов экрана	Значения ldpi, mdpi, hdpi, nodpi, соответствующие 120, 160, 240
Тип сенсорного экрана	Значения finger, notouch, stylus
Клавиатура	Состояние клавиатуры. Значения keysexposed, keyshidden, keyssoft
Текстовый ввод	Значения nokeys, qwerty, 12key (числовое)
Навигация без сенсорного экрана	Значения dpad, nonav, trackball, wheel
Версия SDK	Значения v4 (SDK 1.6), v5 (SDK 2.0) и т. д.

Более подробно эти квалификаторы описаны на следующем сайте: http://developer.android.com/guide/topics/resources/resources-i18n.html#table2.

Перечисленные выше квалификаторы могут использоваться в разнообразных комбинациях, чтобы устройство работало в соответствии с вашими пожеланиями. В названии каталога ресурсов в качестве значения каждого из этих квалификаторов будут использоваться ноль или единица, которые обычно разделяются дефисами. Например, ниже приведено технически правильное название каталога отрисовываемых объектов (правда, использовать такие названия не рекомендуется):

```
drawable-mcc310-en-rUS-large-long-port-mdpi-stylus-keyssoft-qwerty-dpad-v3
```

а вот такой вариант использовать можно:

```
drawable-en-rUS-land (изображения с использованием американского английского, горизонтальный формат)
values-fr (строки по-французски)
```

Независимо от того, сколько квалификаторов ресурсов вы используете в своем приложении, помните, что в коде все ссылки на ресурсы по-прежнему должны иметь вид `R.resource_type.name` — без каких-либо квалификаторов. Например, если вы используете много вариантов файла шаблона `main.xml` в нескольких различных квалифицированных каталогах ресурсов, ваш код по-прежнему будет связан ссылками с `R.layout.main`. Android сама найдет для вас нужный файл `main.xml`.

Адаптеры

В этом разделе мы поговорим об адаптерах. Они выполняют несколько функций, но если говорить в общих чертах, адаптеры упрощают связывание данных с элементом управления, делают этот процесс более гибким. Адаптеры в Android ис-

пользуются при работе с виджетами, дополняющими android.widget.AdapterView. К числу классов, дополняющих AdapterView, относятся ListView, GridView, Spinner и Gallery (рис. 4.20). Сам AdapterView дополняет android.widget.ViewGroup, то есть ListView, GridView и другие элементы управления, являющиеся контейнерами. Иными словами, они отображают коллекцию дочерних элементов управления.

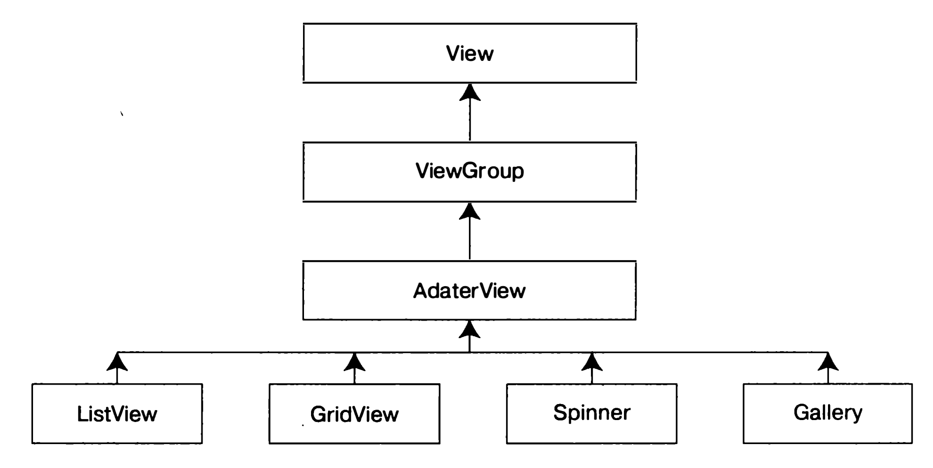


Рис. 4.20. Иерархия класса AdapterView

Назначение адаптера заключается в том, чтобы предоставлять дочерние виды для контейнера. Адаптер берет данные и метаданные определенного контейнера и строит каждый дочерний вид. Рассмотрим этот процесс на практике на примере SimpleCursorAdapter.

Знакомство с SimpleCursorAdapter

Адаптер SimpleCursorAdapter, которым мы уже не раз пользовались, схематически изображен на рис. 4.21.

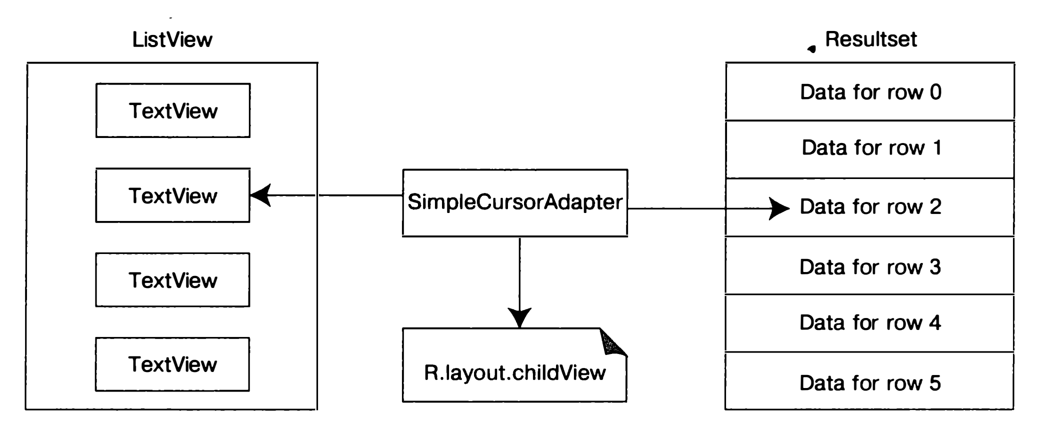


Рис. 4.21. SimpleCursorAdapter

Конструктор SimpleCursorAdapter выглядит следующим образом: SimpleCursorAdapter(Context context, int layout, Cursor c, String[] from, int[] to). Этот адаптер преобразует строку курсора в вид, используемый в качестве дочернего вида в элементе управления контейнере. Определение дочернего вида производится в ресурсе XML (параметр layout). Обратите внимание: поскольку строка, находящаяся в курсоре, может содержать несколько столбцов, следует сообщить SimpleCursorAdapter, какие столбцы необходимо выделить из строки, и для этого создается массив названий столбцов (в котором используется параметр form).

Кроме того, поскольку каждый выделенный вами столбец сопоставляется с информацией из TextView, вы должны указать ID в параметре to. Поскольку существует однозначное соответствие между выделяемым столбцом и TextView, отображающим данные этого столбца, параметры from и to должны иметь одинаковые размеры.

Из рис. 4.21 в некотором отношении становится понятна гибкость, которая приобретается при работе с адаптерами. Поскольку элемент управления контейнер работает с адаптером, можно заменять различные адаптеры в зависимости от используемых данных и дочернего вида. Например, если вы не собираетесь заполнять AdapterView информацией из базы данных, не обязательно использовать SimpleCursorAdapter. Можете остановиться на еще более простом адаптере — ArrayAdapter.

Знакомство с ArrayAdapter

ArrayAdapter — это простейший адаптер, применяемый в Android. Он специально предназначен для работы с элементами списками и всегда предполагает, что в элементе управления TextView содержится список (дочерних видов). Обычно ArrayAdapter создается так:

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
this,android.R.layout.simple_list_item_1,
new string[]{"sayed","satya"});
```

В этом коде конструктор создает адаптер ArrayAdapter, в котором данные элемента TextView представлены в виде строк. Обратите внимание: android.R.layout.simple_list_item_1 указывает на TextView, определенный в инструментарии для разработки в Android (Android SDK).

ArrayAdapter предоставляет удобный метод, которым можно пользоваться, если вы берете данные из файла ресурса. В листинге 4.34 показан соответствующий пример.

Листинг 4.34. Создание ArrayAdapter из строкового файла ресурса

```
Spinner s2 = (Spinner) findViewById(R.id.spinner2);

adapter = ArrayAdapter.createFromResource(this,
R.array.planets,android.R.layout.simple_spinner_item);

adapter.setDropDownViewResource
```

```
                (android.R.layout.simple_spinner_dropdown_item);

s2.setAdapter(adapter);

<string-array name="планеты">
   <item>Mercury</item>
   <item>Venus</item>
   <item>Earth</item>
   <item>Mars</item>
   <item>Jupiter</item>
   <item>Saturn</item>
   <item>Uranus</item>
   <item>Neptune</item>
</string-array>
```

Из листинга 4.34 видно, что в ArrayAdapter используется вспомогательный метод, называемый createFromResource(). Он может создать ArrayAdapter, источник данных для которого будет определяться в строковом файле ресурса. При использовании этого метода вы можете не только извлечь содержимое списка в XML-файл, но и применять локализованные версии.

Создание пользовательских адаптеров

Работать с адаптерами в Android несложно, но они имеют некоторые ограничения. Для решения таких проблем в Android существует абстрактный класс, называемый BaseAdapter, который вы можете при необходимости добавить к собственному адаптеру. Все адаптеры, содержащиеся в Android SDK, дополняют этот базовый адаптер. Следовательно, если вам требуется дополнить адаптер, рассмотрите такие варианты:

- ArrayAdapter<T> — этот адаптер занимает верхнюю позицию в типичном массиве произвольных объектов. Он предназначен для работы с ListView;
- CursorAdapter — также предназначен для работы с ListView; предоставляет данные для списка через курсор;
- SimpleAdapter — как понятно из названия, это очень простой адаптер. Он обычно используется для заполнения списка статическими данными (которые могут быть взяты из ресурсов);
- ResourceCursorAdapter — этот адаптер дополняет CursorAdapter и может создавать виды из ресурсов;
- SimpleCursorAdapter — этот адаптер дополняет ResourceCursorAdapter и создает виды TextView/ImageView из столбцов, содержащихся в курсоре. Виды определяются в ресурсах.

На этом мы завершаем разговор о создании пользовательских интерфейсов. В следующем разделе мы изучим инструмент, предназначенный для просмотра иерархии (Hierarchy Viewer Tool). Он используется для оптимизации и отладки пользовательских интерфейсов.

Отладка и оптимизация шаблонов при помощи инструмента просмотра иерархии

В состав Android SDK входит ряд инструментов, которые позволяют сильно упростить процесс разработки. Поскольку мы говорим о разработке пользовательских интерфейсов, рассмотрим инструмент, предназначенный для просмотра иерархии объектов. Этот инструмент обеспечивает отладку пользовательских интерфейсов на уровне шаблонов (рис. 4.22).

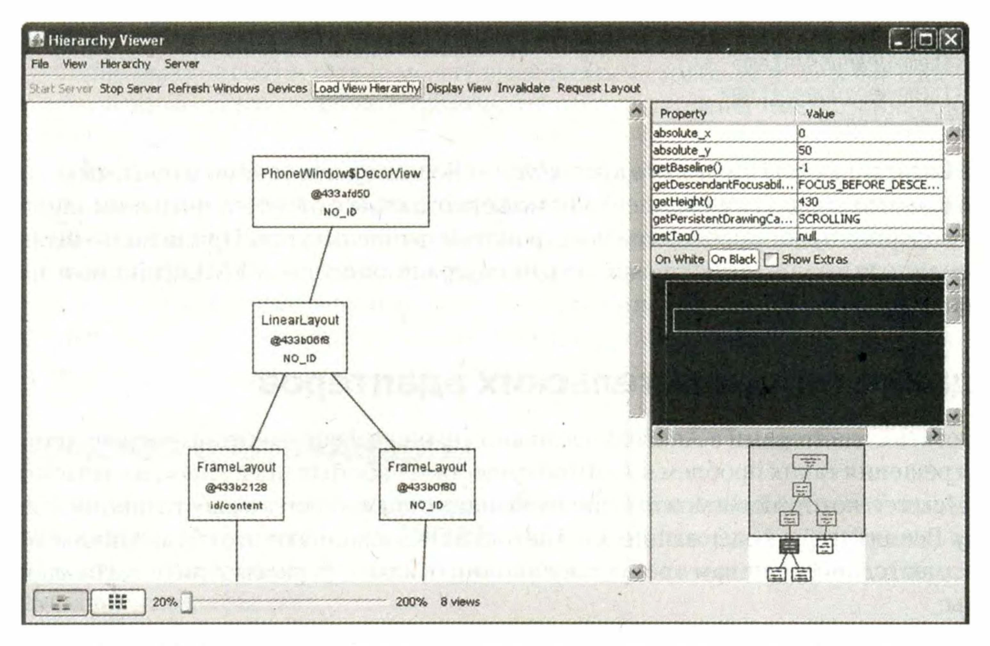


Рис. 4.22. Работа с шаблоном в программе для просмотра иерархии объектов

Из рис. 4.22 понятно, что инструмент для просмотра иерархии отображает иерархию видов в форме древовидной структуры. Работа происходит следующим образом: вы загружаете шаблон в инструмент, а затем проверяете шаблон, чтобы определить возможные недоработки и/или оптимизировать шаблон так, чтобы свести к минимуму количество содержащихся в нем видов (поскольку так можно ускорить работу программы).

Чтобы приступить к отладке пользовательского интерфейса, запустите приложение в эмуляторе и откройте в нем шаблон, который собираетесь оптимизировать. Затем перейдите в каталог `Android SDK /tools`, откуда запускается инструмент для просмотра иерархии. Если программа установлена в Windows, найдите пакетный файл `hierarchyviewer.bat` в каталоге `/tools`. После запуска пакетного файла откроется окно для просмотра иерархии устройств, показанное на рис. 4.23.

В левой области Devices (Устройства) отображается набор устройств (в данном случае — эмуляторов), работающих на машине. При выборе устройства в правой области отображается список его окон. Чтобы просмотреть иерархию видов для отдельно взятого окна, выберите это окно в правой области (обычно названием

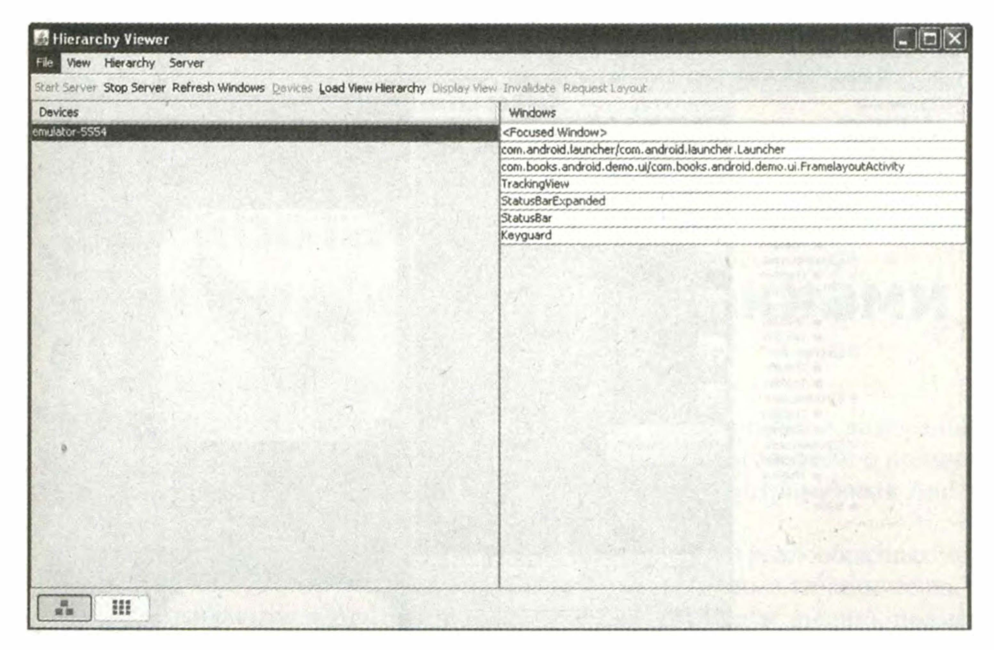


Рис. 4.23. Окно для просмотра иерархии устройств

окна служит полное название явления, перед которым в качестве префикса ставится название пакета прикладных программ), затем нажмите кнопку Load View Hierarchy (Загрузить иерархию видов).

В левой области окна просмотра иерархии видов вы видите иерархию видов конкретного окна (см. рис. 4.22). При выборе элемента вида в левой области можно просмотреть свойства этого вида, открывающиеся справа, а также расположение вида относительно других видов — такое расположение демонстрируется в контурной области (wire-frame pane), находящейся справа. Вокруг выбранного вида появляется красная рамка. Если вы сможете опробовать все виды на практике, то, возможно, найдете способ уменьшить количество видов и, следовательно, ускорить работу приложения.

В нижнем левом углу окна инструмента для просмотра иерархии видов расположены две кнопки (см. рис. 4.22). Левая кнопка позволяет отобразить древовидную структуру видов — этот вариант был рассмотрен выше. Правая кнопка дает возможность показать актуальный шаблон в режиме Pixel Perfect (Идеальная проекция). В таком режиме вы получаете представление шаблона с точностью до пиксела (рис. 4.24). В этом окне есть еще несколько интересных элементов. Слева расположена панель навигации, на которой можно просмотреть все компоненты окна. Если щелкнуть на одном из компонентов, он отобразится в центре, окруженный красной границей. Крестик в середине центральной панели — это прицел, который позволяет увеличить фрагмент изображения, находящегося в центре, и показать такой увеличенный фрагмент в правой области окна (этот инструмент называется Loupe (Лупа)). Элемент Zoom (Масштаб) дает возможность достичь еще более сильного увеличения, чем лупа. Лупа также показывает точное расположение в координатах (x; y) выделенного пиксела и его цветовое значение.

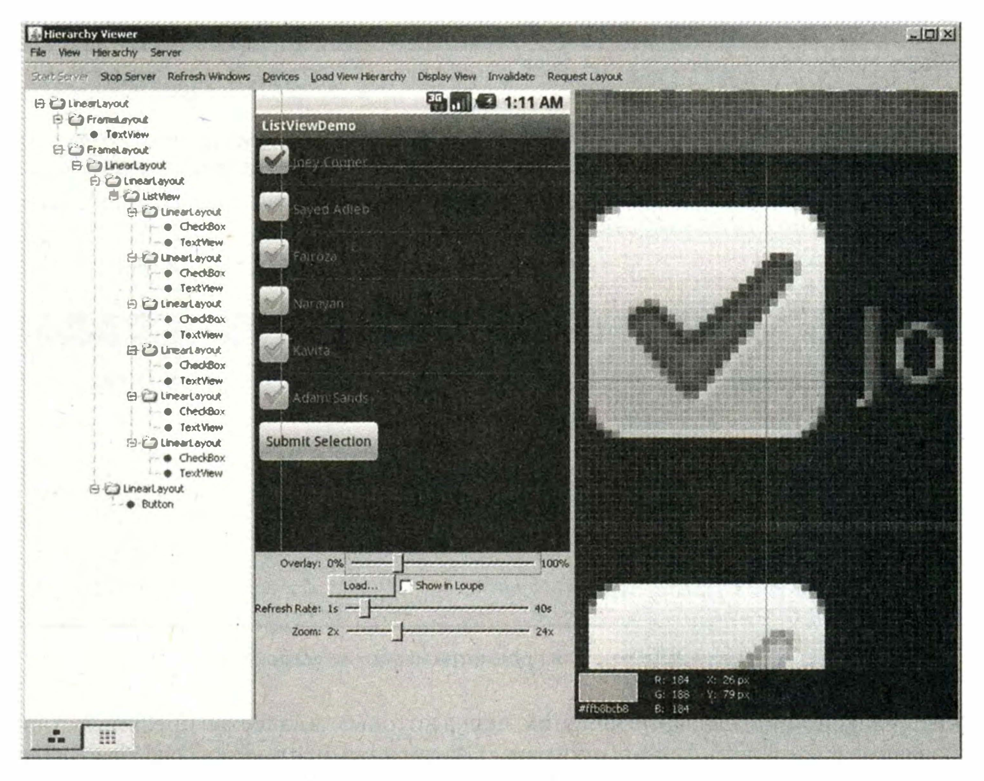


Рис. 4.24. Режим идеальной проекции (Pixel Perfect) инструмента для просмотра иерархии

Необходимо упомянуть еще о двух интересных компонентах данного окна — о кнопке Load (Загрузить) и ползунке Overlay (Наложение). Можно загрузить файл-изображение как фон актуального экрана (в качестве такого фона может использоваться, например, макет экрана, который вы в данный момент разрабатываете). Ползунок Overlay (Наложение) применяется, чтобы увеличивать или уменьшать изображение. Изображение «прикрепляется» к экрану за верхний левый угол. По умолчанию изображение не показывается в лупе, для этого нужно поставить специальный флажок. Имея в арсенале такие инструменты, вы сможете в мельчайших деталях определять внешний вид вашего приложения.

Резюме

На данном этапе вы уже должны хорошо представлять, какие элементы управления есть в инструментарии для разработки в Android. Кроме того, вы должны знать, что представляют собой диспетчеры шаблонов Android и соответствующие адаптеры. Учитывая потенциальные требования к компоновке, связанные с размером экрана, вы должны уметь быстро определять, какие именно элементы управления и диспетчеры шаблонов будут использоваться при разработке наполнения экрана.

В следующей главе мы рассмотрим вопросы разработки пользовательских интерфейсов более детально — поговорим о меню и диалоговых окнах.

5 Работа с меню и диалоговыми окнами

В главе 3 мы начали изучать ресурсы, поставщики содержимого и намерения — базовые компоненты Android SDK. Затем, в главе 4, мы поговорили о пользовательских интерфейсах и шаблонах. В этой главе мы рассмотрим, как в Android функционируют меню и диалоговые окна.

В Android SDK предоставляется комплексная поддержка разнообразных меню и диалоговых окон. В этой главе мы поработаем с несколькими типами меню, которые поддерживаются в Android: с обычными меню (regular menus), подменю (submenu), контекстными меню (context menus), пиктографическими меню (icon menus), вспомогательными меню (secondary menus) и альтернативными меню (alternative menus).

В Android меню являются особым видом ресурсов. Поскольку меню — это ресурсы, в Android их можно загружать из XML-файлов. Android генерирует ID ресурса для каждого из загруженных элементов меню. В этой главе мы подробно рассмотрим такие XML-ресурсы меню. Мы научим вас пользоваться генерируемыми автоматически ID ресурсов с элементами меню любых типов.

Затем мы займемся диалоговыми окнами. Диалоговые окна в Android являются асинхронными, и поэтому работа с ними организована очень гибко. Если вы имеете опыт программирования для такого фреймворка, в котором диалоговые окна могут быть синхронными (например, Microsoft Windows), асинхронные диалоговые окна, возможно, покажутся вам немного непонятными и неудобными. Мы рассмотрим основы создания и использования диалоговых окон в Android и покажем простую и понятную абстракцию, которая поможет вам в работе с диалоговыми окнами.

Меню в Android

Если вам доводилось работать со Swing в Java, с Windows Presentation Foundation (WPF) в Windows или с любым другим фреймворком, предназначенным для создания пользовательских интерфейсов, значит, у вас есть опыт работы с меню. Кроме всесторонней поддержки обычных меню, в Android имеются и некоторые новые шаблоны меню, в частности XML-меню и альтернативные меню.

Сначала мы опишем базовые классы, входящие в фреймворк меню Android. Потом на протяжении главы мы научимся создавать меню и их элементы, а также

рассмотрим, как происходит отклик на элементы меню. Основной класс Android, отвечающий за поддержку меню, — android.view.Menu. Любое явление в Android связано с объектом-меню из этого класса. Такой объект может содержать ряд элементов меню и подменю. Элементы меню представлены в android.view.MenuItem, а подменю — в android.view.SubMenu. Данные отношения схематически изображены на рис. 5.1. Строго говоря, это не диаграмма классов, а структурная схема, составленная для того, чтобы вам было проще представить взаимосвязи между различными относящимися к меню классами и функциями.

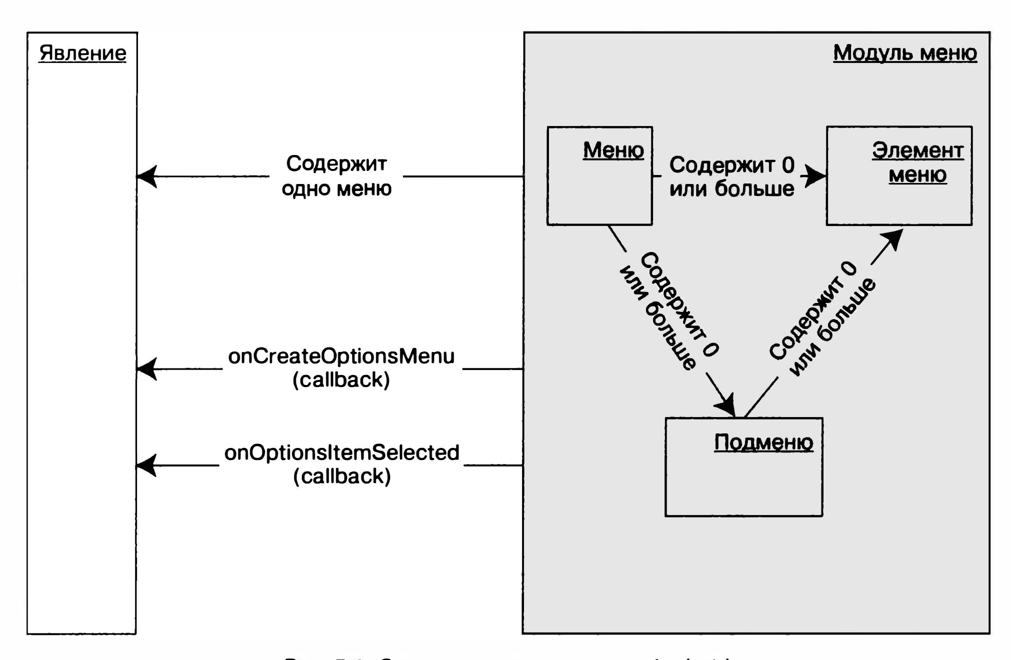


Рис. 5.1. Структура классов меню в Android

Элементы меню можно объединять в группы, присваивая каждой из них групповой ID. ID — это просто атрибут. Несколько элементов меню, имеющие одинаковый групповой ID, считаются элементами одной и той же группы. Кроме группового ID, элемент меню также имеет имя (заголовок), ID элемента меню и ID сортировки (или номер). ID сортировки используется для того, чтобы указать порядок расположения элементов внутри меню. Например, если один из элементов меню имеет ID сортировки 4, а другой элемент меню — ID сортировки 6, то первый элемент будет находиться в меню выше, чем второй.

Некоторые диапазоны номеров сортировки зарезервированы для меню особых видов. Вспомогательные элементы меню, которые считаются менее приоритетными, чем другие, начинаются с 0x30000 и определяются константой Menu.CATEGORY_SECONDARY. Другие типы категорий меню — в частности, системные меню, альтернативные меню и меню-контейнеры — имеют различные диапазоны номеров сортировки. Номера элементов системных меню начинаются с 0x20000 и определяются константой Menu.CATEGORY_SYSTEM. Номера элементов альтернативных меню начинаются с 0x40000 и определяются константой Menu.CATEGORY_ALTERNATIVE. Номера элементов меню-

контейнеров начинаются с 0x10000 и определяются константой Menu.CATEGORY_CONTAINER. Если вы обратите внимание на значения этих констант, то будет понятен и порядок, в котором элементы появляются в меню (различные типы элементов меню будут рассмотрены в разделе «Работа с меню других типов» данной главы).

На рис. 5.1 также показаны два метода обратного вызова, которые можно использовать для создания элементов меню и отклика на них: onCreateOptionsMenu и onOptionsItemSelected. Эти методы будут рассмотрены в следующих подразделах.

Создание меню

В Android SDK вам не требуется создавать объект меню с нуля. Поскольку целое явление соответствует целому меню, Android создает для конкретного явления отдельное меню и передает его методу обратного вызова onCreateOptionsMenu класса явления. (Как понятно из названия метода, буквально означающего «СозданиеМенюПараметров», меню в Android также называются *меню параметров*). Данный метод также позволяет заполнить отдельно взятое переданное меню набором элементов меню (листинг 5.1).

Листинг 5.1. Пример метода onCreateOptionsMenu

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   // вставить элементы меню
   .....
   ...return true;
}
```

После того как меню будет заполнено элементами, код должен вернуть true, чтобы меню было видимым. Если метод возвращает false, меню является невидимым. В коде листинга 5.2 показано, как добавить три элемента меню, используя групповой ID, а также инкрементные ID элементов меню и ID сортировки.

Листинг 5.2. Добавление элементов меню

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   // вызов базового класса для включения системных меню
   super.onCreateOptionsMenu(menu);

   menu.add(0 // Группа
      ,1 // id элемента
      ,0 // порядок
      ,"append"); // заголовок

   menu.add(0,2,1,"item2");
   menu.add(0,3,2,"clear");

   // Чтобы меню было видимым, должно быть возвращено значение true.
   return true;
}
```

Вам следует также вызвать реализацию данного метода в базовом классе, чтобы система могла заполнить меню системными элементами. Чтобы такие системные элементы отделялись от элементов меню других видов, Android добавляет их под номерами, начинающимися с 0x20000. (Как было указано выше, константа Menu.CATEGORY_SYSTEM определяет начальный ID для системных элементов меню.)

Первый параметр, требуемый для добавления элемента меню, — это групповой ID (натуральное число). Второй параметр — это ID элемента меню, который отправляется функции обратного вызова при выборе данного элемента меню. Третий аргумент представляет собой ID порядка сортировки.

Последний аргумент — это имя (заголовок) элемента меню. Его можно записать не обычным текстом, а использовать строковый ресурс посредством файла констант R.java. Групповой ID, ID элемента меню и ID порядка сортировки указывать в таком случае не обязательно. Если вы не хотите указывать все эти ID, используйте Menu.NONE.

Работа с группами меню

Теперь давайте рассмотрим, как строится работа с группами меню. В листинге 5.3 показано, как добавить две группы меню: группу 1 и группу 2.

Листинг 5.3. Использование групповых ID при создании групп меню

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   // группа 1
   int group1 = 1;
   menu.add(group1,1,1,"g1.item1");
   menu.add(group1,2,2,"g1.item2");

   // группа 2
   int group2 = 2;
   menu.add(group2,3,3,"g2.item1");
   menu.add(group2,4,4,"g2.item2");

   return true; // важно вернуть true
}
```

Обратите внимание: ID элементов меню и ID сортировки не зависят от групп. Так что же такое группа? Дело в том, что в Android имеются методы, функционирующие на базе групповых ID. При помощи следующих методов можно управлять элементами меню, входящих в группу:

```
removeGroup(id)
setGroupCheckable(id, checkable, exclusive)
setGroupEnabled(id,boolean enabled)
setGroupVisible(id,visible)
```

Метод removeGroup удаляет все элементы меню из группы с указанным ID. Вы можете активировать или отключать элементы меню в указанной группе, пользуясь

методом setGroupEnabled. Подобным образом можно отображать или скрывать элементы меню определенной группы, применяя метод setGroupVisible.

Метод setGroupCheckable более интересен. Его можно использовать для отображения контрольной отметки (флажка) рядом с выделенным элементом. Если этот метод применяется к группе, данная функция распространяется на все элементы, входящие в группу. Если установить для этого метода исключающую отметку (exclusive flag), то одновременно можно будет выделить только один элемент, входящий в эту группу. Остальные элементы меню отмечены не будут.

Теперь вы знаете, как заполнить основное меню явления набором элементов меню и сформировать из них группы в зависимости от функций тех или иных элементов. Далее мы покажем, как осуществляется отклик на те или иные элементы меню.

Отклик на элементы меню

В Android существует несколько способов отклика на щелчки пользователя на элементах меню. Можно применять метод onOptionsItemSelected класса явления, отдельных слушателей или намерения. Рассмотрим все три способа.

Отклик на элементы меню при помощи onOptionsItemSelected

При нажатии элемента меню Android задействует метод обратного вызова onOptionsItemSelected, относящийся к классу Activity (листинг 5.4).

Листинг 5.4. Пример и основная часть метода onOptionsItemSelected

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
   switch(item.getItemId()) {

   }
   // для обработанных элементов
   return true;

   // для оставшейся части
   ...return super.onOptionsItemSelected(item);
}
```

В данном случае при помощи метода getItemId() класса MenuItem проверяется ID элемента меню и осуществляется необходимая операция. Если onOptionsItemSelected() обрабатывает элемент меню, возвращается значение true. Событие меню больше никуда не передается. Если встречаются обратные вызовы элементов меню, с которыми не может работать onOptionsItemSelected(), он должен вызвать родительский метод при помощи super.onOptionsItemSelected. Применяемая по умолчанию реализация метода onOptionsItemSelected() возвращает false, чтобы могла начаться обычная обработка. При обычной обработке задействуются альтернативные способы активации отклика на нажатия элементов меню.

Отклик на элементы меню при помощи слушателей

Обычно для отклика на щелчок на элементе меню требуется переопределить onOptionsItemSelected; такой метод рекомендуется для повышения эффективности работы. Однако элемент меню позволяет зарегистрировать слушатель, который можно будет использовать для обратного вызова.

Такой процесс выполняется в два этапа. На первом задействуется интерфейс OnMenuClickListener. Затем берется экземпляр реализованного таким образом интерфейса и передается элементу меню. После нажатия элемента меню этот элемент вызывает метод onMenuItemClick() интерфейса OnMenuClickListener (листинг 5.5).

Листинг 5.5. Использование слушателя для обратного вызова, посылаемого в ответ на нажатие элемента меню

```
// этап 1
public class MyResponse implements OnMenuClickListener
{
   // локальная переменная, которая будет использоваться при работе
   //...
   // конструкторы
   @override
   boolean onMenuItemClick(MenuItem item)
   {
      // делаем наши дела
      return true;
   }
}

// этап 2
MyResponse myResponse = new MyResponse(...);
menuItem.setOnMenuItemClickListener(myResponse);
```

Метод onMenuItemClick вызывается при активации какого-либо элемента меню. Этот код выполняется сразу после нажатия элемента, еще до вызова метода onOptionsItemSelected. Если onMenuItemClick возвращает true, больше никаких обратных вызовов не выполняется — в том числе не производится метод onOptionsItemSelected. Это означает, что код слушателя имеет приоритет над методом onOptionsItemSelected.

Отклик на элементы меню при помощи намерений

Вы можете также ассоциировать элемент меню с намерением, применив метод setIntent(intent), относящийся к MenuItem. По умолчанию с элементом меню не связаны никакие намерения. Но если намерение *уже* ассоциировано с элементом меню и, кроме этого намерения, элемент меню больше ничем не обрабатывается, то по умолчанию для активации намерения используется метод startActivity(intent). Чтобы этот процесс функционировал, все обработчики — в частности, метод onOptionsItemSelected — должны вызывать метод onOptionsItemSelected() из родительского класса для обработки тех элементов, которые не ассоциированы с намерениями. Ситуацию можно представить следующим образом: система позволяет onOptionsItemSelected первым приступить к работе с элементами меню (разумеется, после того, как свою работу сделают слушатели).

Если не переопределить метод onOptionsItemSelected, то базовый класс фреймворка Android выполнит всю работу, которая необходима для активации намерения и последующей обработки элемента меню. Но если этот метод будет переопределен, а вас в данный момент не интересует конкретный элемент меню, потребуется вызвать родительский метод, который, в свою очередь, осуществит активацию намерения. В итоге мы либо не переопределяем метод onOptionsItemSelected, либо все же переопределяем его и активируем родительский метод, который будет заниматься откликами от тех элементов меню, которыми в настоящее время не занимаетесь вы.

Создание средства для тестирования меню

Пока все было достаточно просто. Мы изучили, как создавать меню и как осуществлять отклики на связанные с ними события при помощи различных обратных вызовов. Теперь покажем пример явления, используя который попрактикуемся в работе с уже изученными API этого меню.

Цель упражнения — написать простое явление, внутри которого будет содержаться текстовый вид, действующий как отладчик. По мере активации меню мы будем выписывать названия и ID конкретных элементов меню в этот текстовый вид. Готовое приложение Menu будет выглядеть так, как пример на рис. 5.2.

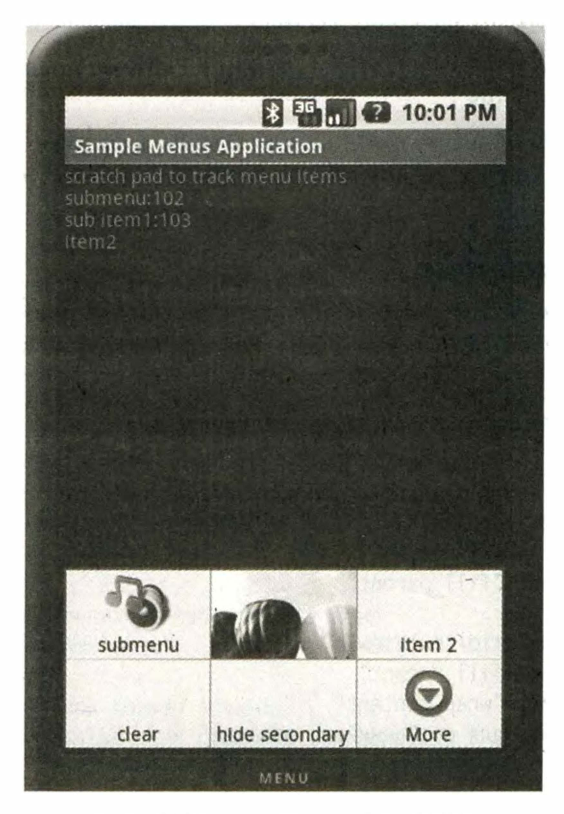


Рис. 5.2. Пример приложения с меню

На рис. 5.2 показаны две интересные вещи: меню и текстовый вид. Меню расположено в нижней части экрана. Однако при запуске приложения вы его не увидите. Чтобы меню открылось, нужно нажать кнопку MENU (Меню) на эмуляторе или устройстве. Еще одна интересная вещь — это текстовый вид, в котором в верхней части экрана выводятся сообщения о ходе отладки. Если нажать элемент меню clear (очистить), программа сотрет все записи с экрана.

ПРИМЕЧАНИЕ

На рис. 5.2 может быть представлен любой этап работы приложения — не обязательно начальный. Мы привели данный рисунок, чтобы проиллюстрировать типы меню, которые будут рассмотрены в этой главе.

Чтобы создать тестовое приложение, выполните следующее.

1. Создайте XML-файл шаблона, в котором будет находиться текстовый вид.
2. Создайте класс Activity, к которому будет относиться шаблон, созданный на шаге 1.
3. Создайте меню.
4. Добавьте в меню несколько обычных элементов.
5. Добавьте в меню в качестве элементов несколько вспомогательных меню.
6. Протестируйте отклик на элементы меню.
7. Измените файл AndroidManifest.xml, чтобы в системе правильно отображался заголовок приложения.

Все эти операции будут подробно рассмотрены в следующих разделах вместе с исходным кодом, который потребуется для написания тестового приложения.

Создание XML-шаблона

На первом этапе создается простой файл шаблона, написанный на XML, в котором расположен текстовый вид (листинг 5.6). Этот файл можно загрузить в явление при запуске.

Листинг 5.6. XML-файл шаблона для тестового приложения

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 >
<TextView android:id="@+id/textViewId"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Блокнот для сообщений об отладке"
 />
</LinearLayout>
```

Создание явления (Activity)

На шаге 2 создается явление — это тоже очень просто. Если файл шаблона расположен в \res\layout\main.xml, как это и должно быть, можно получить доступ к этому файлу через его ID, а затем заполнить текстовый вид информацией (листинг 5.7).

Листинг 5.7. Класс Activity из тестового приложения

```
public class SampleMenusActivity extends Activity {

   // инициализация этого кода в onCreateOptions
   Menu myMenu = null;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.main);
}
```

Для краткости мы исключили из этого кода операторы импорта (import statements). В Eclipse можно автоматически вставить операторы импорта, перетянув контекстное меню в редактор и выбрав **Source ~TRA Organize Imports** (Источник ~ TRA-организация импорта).

Настройка меню

Теперь у нас есть и вид, и явление, и мы переходим к шагу 3: переопределяем onCreateOptionsMenu и настраиваем меню программными средствами (листинг 5.8).

Листинг 5.8. Настройка меню программными средствами

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   // вызов родительского метода
   // для прикрепления меню с любых уровней системы
   super.onCreateOptionsMenu(menu);

   this.myMenu = menu;

   // добавление нескольких обычных меню
   addRegularMenuItems(menu);

   // добавление нескольких вспомогательных меню
   add5SecondaryMenuItems(menu);

   // для показа меню код должен возвращать true
   // при возвращении false меню отображаться не будет
   return true;
}
```

Код из листинга 5.8 вызывает сначала родительский метод onCreateOptionsMenu, чтобы он мог добавлять меню с любых уровней системы.

ПРИМЕЧАНИЕ

Во всех имеющихся версиях Android SDK метод onCreateOptionsMenu не добавляет в меню новые элементы. Однако в будущих версиях такая функция может появиться — поэтому и рекомендуется вызывать родительский метод.

После этого код запоминает объект Menu, чтобы оперировать им позже в демонстрационных целях. После этого код продолжает работу и добавляет несколько обычных и несколько вспомогательных элементов меню.

Добавление обычных элементов меню

Переходим к шагу 4: добавляем в меню несколько обычных элементов. Код для addRegularMenuItems показан в листинге 5.9.

Листинг 5.9. Функция addRegularMenuItems

```
private void addRegularMenuItems(Menu menu)
{
   int base=Menu.FIRST; // значение 1

   menu.add(base,base,base,"append");
   menu.add(base,base+1,base+1,"item 2");
   menu.add(base,base+2,base+2,"clear");

   menu.add(base,base+3,base+3,"hide secondary");
   menu.add(base,base+4,base+4,"show secondary");

   menu.add(base,base+5,base+5,"enable secondary");
   menu.add(base,base+6,base+6,"disable secondary");

   menu.add(base,base+7,base+7,"check secondary");
   menu.add(base,base+8,base+8,"uncheck secondary");
}
```

В классе Menu определяется несколько констант, облегчающих работу, одна из них — Menu.FIRST. Ее можно использовать как исходный номер для ID разных меню и при другой последовательной нумерации, относящейся к меню. Посмотрите, как можно сосредоточить действие группового ID на base и производить приращение только ID порядка сортировки и ID элементов меню. Кроме того, в коде содержится несколько специфических элементов меню, в частности "hide secondary,", "enable secondary," и т. д., демонстрирующих некоторые концепции, связанные с меню.

Добавление вспомогательных элементов меню

Для выполнения шага 5 потребуется добавить несколько вспомогательных элементов меню (листинг 5.10). Выше мы говорили о том, что нумерация вспомогательных элементов меню начинается с 0x30000 и определяется в константе Menu.CATEGORY_SECONDARY. ID порядка сортировки, применяемые с такими элементами меню, выше, чем аналогичные ID обычных элементов, поэтому вспомогательные элементы ото-

бражаются в меню после обычных. В остальном работа вспомогательного элемента меню строится так же, как и работа обычного элемента.

Листинг 5.10. Добавление вспомогательных элементов меню

```
private void add5SecondaryMenuItems(Menu menu)
{
 // вспомогательные элементы меню отображаются так же, как и любые другие
 int base=Menu.CATEGORY_SECONDARY;

 menu.add(base,base+1,base+1,"sec. item 1");
 menu.add(base,base+2,base+2,"sec. item 2");
 menu.add(base,base+3,base+3,"sec. item 3");
 menu.add(base,base+3,base+3,"sec. item 4");
 menu.add(base,base+4,base+4,"sec. item 5");
}
```

Реагирование на нажатия элементов меню

Теперь меню настроены, и мы приступаем к шагу 6: программируем отклик на нажатие этих элементов. Если пользователь нажимает элемент меню, Android задействует метод обратного вызова onOptionsItemSelected класса Activity, передавая ссылку на тот элемент меню, с которым взаимодействовал пользователь. Затем к MenuItem применяется метод getItemId(), позволяющий узнать, какой именно элемент это был.

Часто в коде встречаются оператор switch либо серии операторов if и else, вызывающих различные функции в ответ на нажатие элементов меню. В листинге 5.11 показан стандартный пример такого кода, предназначенного для отклика на нажатия элементов меню, с применением метода обратного вызова onOptionsItemSelected. (Немного ниже мы изучим, как та же операция выполняется при помощи загрузки меню из XML-файлов, где мы будем использовать вместо ID элементов меню символические имена.)

Листинг 5.11. Отклики на нажатия элементов меню

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   if (item.getItemId() == 1)     {
      appendText("\nhello");
   }
   else if (item.getItemId() == 2) {
      appendText("\nitem2");
   }
   else if (item.getItemId() == 3) {
      emptyText();
   }
   else if (item.getItemId() == 4) {
      // спрятать вспомогательный элемент
      this.appendMenuItemText(item);
      this.myMenu.setGroupVisible(Menu.CATEGORY_SECONDARY,false);
   }
   else if (item.getItemId() == 5) {
```

```
        // показать вспомогательный элемент
        this.appendMenuItemText(item);
        this.myMenu.setGroupVisible(Menu.CATEGORY_SECONDARY,true);
    }
    else if (item.getItemId() == 6) {
        // задействовать вспомогательный элемент
        this.appendMenuItemText(item);
        this.myMenu.setGroupEnabled(Menu.CATEGORY_SECONDARY,true);
    }
    else if (item.getItemId() == 7) {
        // деактивировать вспомогательный элемент
        this.appendMenuItemText(item);
        this.myMenu.setGroupEnabled(Menu.CATEGORY_SECONDARY,false);
    }
    else if (item.getItemId() == 8) {
        // отметить вспомогательный элемент
        this.appendMenuItemText(item);
        myMenu.setGroupCheckable(Menu.CATEGORY_SECONDARY,true,false);
    }
    else if (item.getItemId() == 9) {
        // снять отметку со вспомогательного элемента
        this.appendMenuItemText(item);
        myMenu.setGroupCheckable(Menu.CATEGORY_SECONDARY,false,false);
    }
    else    {
        this.appendMenuItemText(item);
    }
    // Должен быть возвращен элемент меню,
    // с которым происходило взаимодействие.
    return true;
}
```

В листинге 5.11 некоторые операции, производимые над меню, осуществляются на уровне группы. Вызовы таких методов выделены полужирным шрифтом. Код также журналирует подробную информацию о нажатом элементе меню в TextView. В листинге 5.12 показаны некоторые вспомогательные функции, которые применяются при записи текста в TextView. Обратите внимание: для получения заголовка MenuItem применяется дополнительный метод.

Листинг 5.12. Вспомогательные функции, предназначенные для записи информации в текстовый вид, используемый при отладке

```
// Прикрепляем имеющуюся текстовую строку к TextView.
 private void appendText(String text) {
     TextView tv = (TextView)this.findViewById(R.id.textViewId);
     tv.setText(tv.getText() + text);
 }

// Прикрепляем имеющуюся текстовую строку к TextView.
 private void appendMenuItemText(MenuItem menuItem) {
```

```
        String title = menuItem.getTitle().toString();
        TextView tv = (TextView)this.findViewById(R.id.textViewId);
            tv.setText(tv.getText() + "\n" + title);
    }
// удаление содержимого из TextView
    private void emptyText() {
        TextView tv = (TextView)this.findViewById(R.id.textViewId);
        tv.setText("");
    }
```

Доработка файла AndroidManifest.xml

Последний этап создания тестового приложения — обновление файла AndroidManifest.xml, относящегося к приложению. Этот файл автоматически создается системой, когда вы начинаете новый проект, и находится в корневом каталоге вашего проекта.

На данном этапе регистрируется класс Activity (например, SampleMenusActivity) и указывается его название. Мы назвали данное явление **Sample Menus Application** (Пример приложения с меню) (см. рис. 5.2). Обратите внимание на запись, выделенную в листинге 5.13.

Листинг 5.13. Файл AndroidManifest.xml для тестового приложения

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="imja-vashego-paketa-pishetsja-zdes „
      android:versionCode="1"
      android:versionName="1.0.0">
   <application android:icon="@drawable/icon" android:label="Образец меню">
      <activity android:name=".SampleMenusActivity"
              android:label="Sample Menus Application">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
   </application>
</manifest>
```

Воспользовавшись показанным здесь кодом, вы сможете быстро написать тестовое приложение, с помощью которого будете экспериментировать с меню. Мы показали, как создать простое явление, которое инициализируется с текстовым видом, как заполнить этот вид и как организовать отклик на меню. Большинство меню работают в соответствии с несложной, но эффективной моделью. С помощью рис. 5.2 вы можете примерно представить, какой пользовательский интерфейс у вас получится после завершения этого упражнения. Но, как уже было указано выше, результат не будет полностью совпадать с рисунком, так как вы еще не научились добавлять пиктографические меню. И даже после добавления пиктографических меню полное совпадение не гарантируется, поскольку вы можете выбрать не те рисунки, которыми пользовались мы.

Работа с меню других типов

Пока мы изучили некоторые простые, но широко используемые типы меню. Углубляясь в работу с SDK, вы увидите, что в Android также поддерживаются пиктографические меню, подменю, контекстные меню и альтернативные меню. Альтернативные меню встречаются только в Android. В данном разделе мы рассмотрим все эти типы меню.

Расширенные меню

Вспомните рис. 5.2, где в меню программы-образца есть элемент More (Еще), расположенный в правом нижнем углу меню. Мы не увидели ни в одном из фрагментов кода, как добавлять в меню такой элемент. Откуда же он берется?

Если в меню приложения входит больше элементов, чем можно отобразить на экране, Android показывает в меню элемент More (Еще), нажав который пользователь увидит все остальные элементы. Такое меню, называемое *расширенным* (expanded), отображается автоматически, если в меню слишком много элементов, и все они не помещаются в имеющемся пространстве. Но с расширенным меню связано одно ограничение — в нем нельзя использовать пиктограммы. Нажав More (Еще), пользователь увидит все дополнительные пункты меню без пиктограмм.

Работа с пиктографическими меню

Мы уже коротко упоминали пиктографические меню. Теперь рассмотрим их более подробно. Android поддерживает в меню не только текст, но и изображения или пиктограммы. Их можно использовать для представления элементов меню — вместо текста или вместе с ним. Однако не забывайте и о некоторых ограничениях, связанных с использованием пиктографических меню. Во-первых, как было сказано в предыдущем подразделе, пиктограммы нельзя использовать в расширенных меню. Во-вторых, элементы пиктографических меню нельзя отмечать флажками (check marks). В-третьих, если текстовое название пиктографического меню является слишком длинным, оно будет обрываться через определенное количество букв, в зависимости от размеров дисплея (последнее ограничение касается и текстовых элементов меню).

Создать пиктографический элемент меню очень просто. Необходимо сделать обычный текстовый элемент меню, потом воспользоваться методом setIcon класса MenuItem, который позволяет задать для данного текста изображение. При этом нужно использовать ID ресурса, присвоенный изображению. Это значит, что такой ID сначала нужно сгенерировать, поместив изображение или пиктограмму в каталог /res/drawable. Например, если файл пиктограммы называется balloons, ID ресурса будет R.drawable.balloons. В следующем примере кода показано, как это делается:

```
// Добавить элемент меню и запомнить его,
// чтобы впоследствии с ним можно было работать.
// Затем ассоциировать его с пиктограммой.
MenuItem item8 = menu.add(base,base+8,base+8,"uncheck secondary");
item8.setIcon(R.drawable.balloons);
```

Когда вы добавляете в меню новые элементы, вам обычно не требуется сохранять локальную переменную, возвращаемую методом menu.add. Но в таком случае системе нужно будет запомнить возвращенный объект, чтобы потом можно было связать пиктограмму с элементом меню. Из кода, приведенного в этом примере, также видно, что метод menu.add возвращает тип MenuItem.

Пиктограмма отобразится, как только элемент появится на основном экране приложения. Если элемент будет показан как часть расширенного меню, то пиктограмма не отобразится — вы увидите только текст. Элемент меню, вместе с которым отображаются нарисованные шарики, — как раз пример использования пиктограммы в таком качестве.

Работа с подменю

Теперь рассмотрим подменю, используемые в Android. На рис. 5.1 показаны структурные отношения подменю (SubMenu), меню (Menu) и элементов меню (MenuItem). Объект Menu может иметь несколько объектов SubMenu. Каждый объект SubMenu добавляется к Menu путем вызова метода Menu.addSubMenu (листинг 5.14). Элементы добавляются в подменю по тому же принципу, что и при работе с меню. Причина в том, что SubMenu также является объектом, производным от Menu. Правда, добавлять в подменю новые подменю вы не сможете.

Листинг 5.14. Добавление подменю

```
private void addSubMenu(Menu menu)
{
   // вспомогательные элементы показаны так же, как и любые другие
   int base=Menu.FIRST + 100;
   SubMenu sm = menu.addSubMenu(base,base+1,Menu.NONE,"submenu");
   sm.add(base,base+2,base+2,"sub item1");
   sm.add(base,base+3,base+3, "sub item2");
   sm.add(base,base+4,base+4, "sub item3");

   // пиктограммы в элементах подменю не поддерживаются
   item1.setIcon(R.drawable.icon48x48_2);

   // но такой код все же является правильным
   sm.setIcon(R.drawable.icon48x48_1);

   // в результате получится исключение времени выполнения
   // sm.addSubMenu("попробуйте это");
}
```

ПРИМЕЧАНИЕ

SubMenu как субкласс объекта Menu также использует метод addSubMenu. Компилятор не будет выдавать предупреждения, если вы добавите в подменю другое подменю, но если вы попытаетесь так сделать, программа выдаст исключение времени исполнения.

В документации по Android SDK также содержится указание на то, что в подменю не могут находиться пиктографические элементы. Если добавить пиктограмму к элементу меню, а затем поместить этот элемент в подменю, данная пиктограмма

будет игнорироваться, даже если система и не сообщит об ошибке времени компиляции или ошибке времени исполнения. Правда, само подменю может обозначаться пиктограммой.

Предпосылки для вставки системных меню

В большинстве приложений Windows используются такие меню, как File (Файл), Edit (Правка), View (Вид), Open (Открыть), Close (Закрыть) и Exit (Выход). Эти меню называются системными. В Android SDK указано, что система может вставить в приложение сходный набор меню, если создано меню параметров. Однако в актуальных версиях Android SDK ни одно из перечисленных меню не закладывается в системе автоматически в процессе создания. Возможно, в следующих версиях такие системные меню будут реализованы. В документации программистам рекомендуется оставлять в коде места для вставки таких меню, чтобы в будущем их можно было добавить. Для этого вам следует вызвать метод onCreateOptionsMenu родительского элемента, который позволяет добавлять системные меню в группу, идентифицируемую константой CATEGORY_SYSTEM.

Работа с контекстными меню

Пользователи ПК, несомненно, не раз встречались с контекстными меню. Например, в приложениях Windows можно открыть контекстное меню, щелкнув правой кнопкой мыши на элементе пользовательского интерфейса. В Android идея контекстных меню реализована при помощи действия, называемого *долгим щелчком* (long click). При долгом щелчке кнопка мыши удерживается на виде Android чуть дольше, чем при обычном щелчке.

На мобильных устройствах, например сотовых телефонах, щелчки кнопкой мыши реализуются несколькими способами, в зависимости от принципа навигации. Если на телефоне есть колесико для передвижения указателя, щелчок делается нажатием этого колесика. Устройство может иметь сенсорную панель, при движении пальцем или нажатии которой будет происходить щелчок. Кроме того, на устройстве может быть специальный набор кнопок — со стрелками для перемещения мыши и кнопкой в центре, при помощи которой и делается щелчок. Независимо от того, как именно щелчок мыши реализован на устройстве, долгий щелчок — это просто удлиненный обычный щелчок.

В структурном отношении контекстное меню отличается от обычных меню параметров, о которых мы уже говорили (рис. 5.3). В контекстных меню есть некоторые нюансы, которых нет в меню параметров.

На рис. 5.3 видно, что в архитектуре меню Android контекстные меню представлены классом ContextMenu. Как и в Menu, в ContextMenu могут содержаться элементы. Для добавления элементов в контекстные меню используются методы, относящиеся к Menu. Основная разница между Menu и ContextMenu сводится к вопросу о принадлежности меню. Обычное меню параметров принадлежит к определенному явлению, а контекстное меню — к виду. И это неудивительно, так как для активации контекстных меню щелкают именно на *виде*. Итак, в явлении может содержаться только одно меню параметров, но много контекстных меню. Ведь

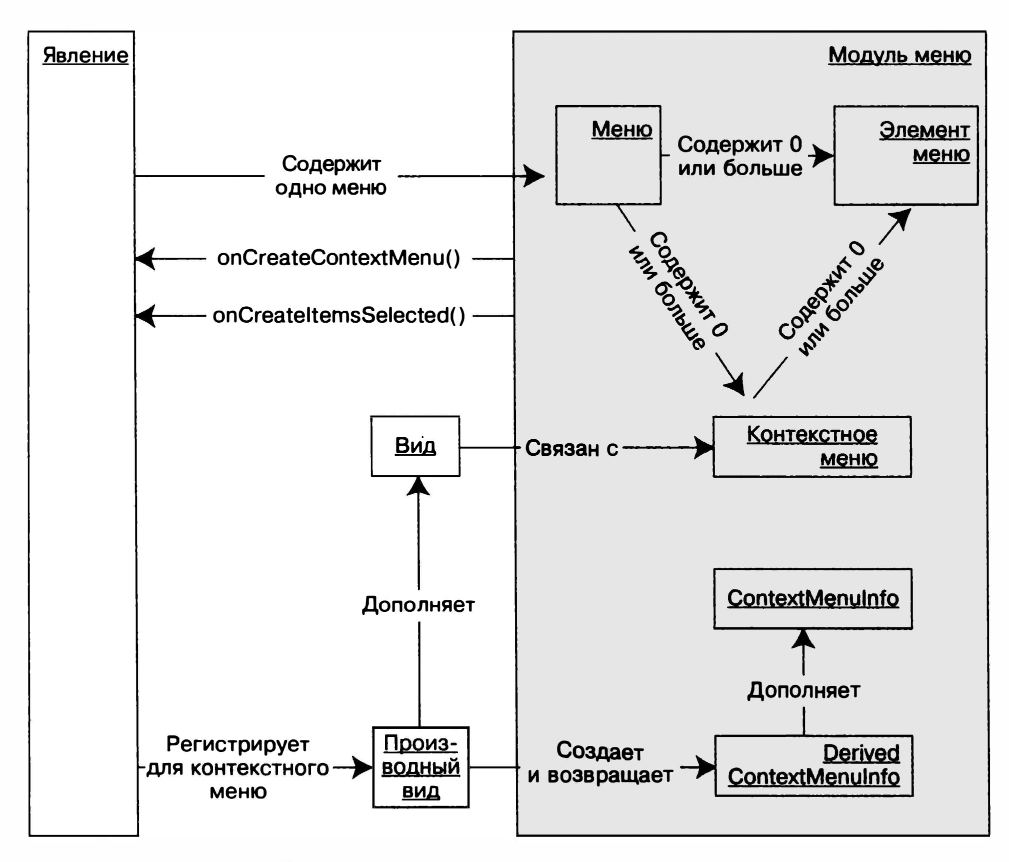


Рис. 5.3. Явления, виды и контекстные меню

в явлении может быть много видов, и каждый вид может иметь собственное контекстное меню. Значит, в явлении может быть столько же контекстных меню, сколько и видов.

Хотя контекстное меню и относится к виду, метод для заполнения контекстных меню находится в классе `Activity`. Этот метод называется `activity.onCreateContextMenu()`, функционально он схож с методом `activity.onCreateOptionsMenu()`. Вместе с этим методом обратного вызова также передается вид, в контекстное меню которого нужно поставить элементы.

Следует отметить еще одну деталь, связанную с контекстными меню. В то время, как метод `onCreateOptionsMenu()` автоматически вызывается для каждого явления, с `onCreateContextMenu()` этого не происходит. Вид, находящийся в явлении, не обязательно *должен* иметь контекстное меню. Например, вы создаете явление с тремя видами, но хотите, чтобы контекстное меню было только у одного вида, а не у всех. Если вы хотите, чтобы в конкретном виде было свое контекстное меню, этот вид нужно специально зарегистрировать как имеющий контекстное меню вместе с явлением этого вида. Это делается при помощи метода `activity.registerForContextMenu(view)`, который мы рассмотрим в подразделе «Регистрация вида при создании контекстного меню» ниже.

Теперь обратите внимание на класс ContextMenuInfo, показанный на рис. 5.3. Объект такого типа передается методу onCreateContextMenu. Это один из способов, которым вид может передать методу дополнительную информацию. Чтобы вид мог это сделать, ему нужно переопределить метод getContextViewInfo() и вернуть производный класс ContextMenuInfo с методами, предназначенными для представления дополнительной информации. Чтобы полностью понять данное взаимодействие, можете изучить исходный код android.view.View.

ПРИМЕЧАНИЕ

Согласно документации Android SDK, в контекстных меню не поддерживаются горячие клавиши, пиктограммы и подменю.

Теперь, когда мы изучили общую структуру контекстных меню, рассмотрим несколько образцов кода, в которых демонстрируются все этапы внедрения контекстных меню в систему.

1. Зарегистрируйте вид, к которому будет относиться контекстное меню, в методе onCreate() определенного явления.
2. Заполните контекстное меню при помощи метода onCreateContextMenu(). Прежде чем Android активирует этот метод обратного вызова, нужно завершить шаг 1.
3. Система начнет отвечать на щелчки по контекстным меню.

Регистрация вида при создании контекстного меню

При внедрении в систему контекстного меню прежде всего нужно зарегистрировать в методе onCreate() определенного явления тот вид, в котором будет содержаться контекстное меню. Если вы воспользуетесь рассмотренным выше в этой главе приложением для тестирования меню, можно зарегистрировать TextView как вид, содержащий контекстное меню. В листинге 5.15 показано, как это сделать. Сначала нужно найти TextView, а затем вызвать метод registerForContextMenu явления, используя TextView как аргумент. После этого в TextView можно будет сделать контекстное меню.

Листинг 5.15. Регистрация вида TextView при создании контекстного меню

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   TextView tv = (TextView)this.findViewById(R.id.textViewId);
   registerForContextMenu(this.getTextView());
}
```

Заполнение контекстного меню

После того как вид (в данном примере — TextView) будет зарегистрирован для создания контекстного меню, Android вызовет метод onCreateContextMenu(), задав этот

вид как аргумент. На этом этапе вы сможете заполнить контекстное меню необходимыми аргументами. Метод обратного вызова onCreateContextMenu() дает три аргумента, которые мы затем используем при работе.

Первый аргумент — это готовый объект ContextMenu, второй — это вид (например, TextView), генерирующий обратный вызов, а третий — это класс ContextMenuInfo, который мы кратко рассмотрели, обсуждая рис. 5.3. При работе со многими простыми классами объект ContextMenuInfo можно просто игнорировать. Однако в некоторых видах с этим объектом может передаваться дополнительная информация. В таких случаях необходимо привести класс ContextMenuInfo к субклассу, а затем применить дополнительные методы, чтобы получить дополнительную информацию.

К классам, производным от ContextMenuInfo, относятся, в частности, AdapterContextMenuInfo и ExpandableContextMenuInfo. Виды, связанные в Android с курсорами базы данных, используют класс AdapterContextMenuInfo, чтобы передавать ID строки в рамках вида, контекстное меню которого в настоящий момент отображается. Это означает, что вы сможете точно указывать объект, на котором был сделан щелчок, а не просто передавать вид, в котором этот элемент находится.

В листинге 5.16 показано, как используется метод onCreateContextMenu().

Листинг 5.16. Метод onCreateContextMenu()

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo menuInfo)
{
 menu.setHeaderTitle("Sample Context Menu");
 menu.add(200, 200, 200, "item1");
}
```

Отклик на элементы контекстного меню

Третий этап создания контекстного меню — организация отклика на щелчки на его элементах. Этот механизм напоминает принцип отклика системы на нажатие элементов обычных меню параметров. В Android существует метод обратного вызова, похожий на onOptionsItemSelected(), — он называется onContextItemSelected(). Данный метод, как и его аналог onOptionsItemSelected(), относится к классу Activity. В листинге 5.17 показан метод onContextItemSelected().

Листинг 5.17. Отклик на нажатия элементов контекстных меню

```
@Override
public boolean onContextItemSelected(MenuItem item)
{
 if (item.itemId() = id-opredelennogo-elementa-menu)
 {
    // обработка данного элемента меню
return true;
 }
… other exception processing
}
```

Работа с альтернативными меню

Вы уже знаете, как создавать меню, подменю и контекстные меню и как с ними работать. В Android существует еще один, новый вид меню — ***альтернативные меню***, элементы которых могут входить в состав меню, подменю или контекстных меню. Альтернативные меню позволяют одному приложению Android пользоваться функциями другого. Такие меню входят в состав фреймворка Android, обеспечивающего внутрисистемный обмен информацией между приложениями и работу с системой.

Итак, альтернативные меню позволяют включать компоненты меню одних приложений в меню других приложений. При выборе альтернативных меню нужное приложение или явление запускается при помощи URL, указывающей на данные, необходимые для работы этого явления. Затем активированное явление использует переданную от намерения URL с данными. Итак, чтобы хорошо понимать работу с альтернативными меню, вы должны разбираться в поставщиках содержимого, URI контента, MIME-типах контента и намерениях (см. главу 3).

Общая идея в данном случае такова: предположим, вы создаете экран, на котором будут отображаться какие-либо данные. Скорее всего, такой экран будет представлять собой явление. В этом явлении у вас будет меню параметров, которое позволит манипулировать с данными при помощи ряда способов. Теперь давайте представим, что мы работаем с документом или записью, которые идентифицированы при помощи URI и соответствующего типа MIME. Как программист, вы в первую очередь предположите, что на устройстве есть не одна, а несколько программ, которые могут работать с такими данными и отображать их. После этого вы попробуете дать группе таких программ возможность отображать элементы своих меню как компоненты меню того явления, которое вы сейчас создаете.

Чтобы прикрепить к обычному меню элементы из альтернативных меню, выполните следующие шаги — при этом меню настраивается в методе onCreateOptionsMenu.

1. Создайте намерение, URI данных которого соответствует URI информации, отображаемой в настоящий момент.
2. Установите для намерения категорию CATEGORY_ALTERNATIVE.
3. Найдите явления, которые позволяют производить операции с данными, поддерживаемыми URI такого типа.
4. Добавьте в качестве элементов меню намерения, которые могут активировать такие явления.

Выполняя эти операции, вы узнаете много важных деталей о природе приложений в Android, поэтому рассмотрим каждый этап подробно. Как вы уже знаете, прикрепление элементов альтернативного меню к обычному меню осуществляется методом onCreateOptionsMenu:

```
@Override public boolean onCreateOptionsMenu(Menu menu)
{
}
```

Разберем код, из которого состоит эта функция. Сначала вам нужно знать URI данных, с которыми, возможно, придется работать в рамках этого явления. Получить URI можно следующим образом:

```
this.getIntent().getData()
```

Такой код работает, поскольку класс Activity имеет метод getIntent(), возвращающий URI с данными, по которому активируется нужное явление. Явление может быть основным и активироваться из главного меню. Тогда намерение может отсутствовать и метод getIntent() вернет null. Не допускайте таких ситуаций в коде.

Теперь мы должны найти другие программы, которые умеют обращаться с данными такого типа. При поиске используем намерение в качестве аргумента. Вот код для создания такого намерения:

```
Intent criteriaIntent = new Intent(null, getIntent().getData());
intent.addCategory(Intent.CATEGORY_ALTERNATIVE);
```

При создании намерения мы также добавляем категорию интересующих нас действий. Если быть точными, нас интересуют только такие явления, которые можно активировать как часть альтернативного меню. И вот теперь мы готовы приказать объекту Menu начать поиск подходящих явлений и добавить подходящие явления в меню в качестве параметров (листинг 5.18).

Листинг 5.18. Заполнение меню элементами из альтернативных меню

```
// поиск подходящих явлений и наполнение ими меню
menu.addIntentOptions(
   Menu.CATEGORY_ALTERNATIVE,   // группа
   Menu.CATEGORY_ALTERNATIVE,   // Любые уникальные ID,
                         // которые может потребоваться добавить
   Menu.CATEGORY_ALTERNATIVE,   // порядок.
   getComponentName(),          // Имя класса, отображающего
                         // меню - здесь, вот этот класс.
   null,                 // никаких особенностей
   criteriaIntent,          // Предварительно созданное намерение,
                         // описывающее наши требования.
   0,                    // без дополнительных отметок
   null);                // возвращенные элементы меню
```

Прежде чем рассмотреть каждую строку этого кода, мы объясним, что такое подходящие явления (matching activities). *Подходящее явление* — это такое явление, которое может обработать данный URI. Обычно информация о том, с какими URI может работать явление, содержится и регистрируется в файле описания этого явления при помощи URI, действий и категорий. В Android имеется механизм, позволяющий использовать объект Intent для поиска подходящих явлений на основании атрибутов (URI, действия и категории).

Теперь подробнее изучим листинг 5.18. Метод addIntentOptions класса Menu отвечает за поиск явлений, подходящих для URI такого намерения, а также поиск атрибутов категории. Затем метод добавляет эти явления в меню, в нужную группу и с нужными ID элементов меню и порядка сортировки. Три первых аргумента касаются этого аспекта работы метода. В листинге 5.18 мы начинаем работу

с Menu.CATEGORY_ALTERNATIVE — это группа, в которую будут добавляться новые элементы меню. Та же константа используется в качестве исходной точки при определении ID элементов меню и ID порядка сортировки.

Следующий аргумент указывает на полностью квалифицированное имя компонента явления. В состав данного явления входит рассматриваемое меню. В коде используется вспомогательный метод, называемый getComponentName(). На примере этого метода вы можете поучиться извлекать имена компонентов из имен классов и пакетов. Имя компонента нужно нам потому, что при добавлении нового элемента меню он должен будет активировать целевое явление. Для осуществления этой операции система должна знать начальное явление, при помощи которого было запущено целевое. Следующий аргумент — это массив намерений, который будет использоваться в качестве фильтра при обработке возвращаемых намерений.

Далее идет аргумент, указывающий на criteriaIntent, который мы только что создали. Это критерии поиска, которыми мы будем пользоваться. За ним идет аргумент, представляющий собой индикатор, например Menu.FLAG_APPEND_TO_GROUP, указывающий, что нужно сделать с этим элементом — добавить его в существующую группу элементов меню или заменить один из элементов этой группы. По умолчанию задается значение 0, при котором новые элементы меню должны заменять те, которые уже имеются в группе.

Последний аргумент в листинге 5.18 — это массив добавленных элементов меню. Вы можете пользоваться ссылками на эти элементы меню, если хотите произвести с ними еще какие-то операции после добавления.

Это, конечно, хорошо, но осталось еще несколько нерешенных вопросов. Например, как будут называться элементы, добавленные в меню? В документации к Android об этом тактично умалчивается. Поэтому нам пришлось перелопатить немало исходного кода, чтобы понять, каким же именно образом эта функция реализуется внутри системы.

Оказалось, что класс Menu — это просто интерфейс, поэтому для него не удалось найти исходного кода реализации (в главе 1 рассказано, где можно ознакомиться с исходным кодом Android). Класс, реализующий интерфейс Menu, называется MenuBuilder. В листинге 5.19 показан исходный код важного метода addIntentOptions, относящегося к классу MenuBuilder. (Код приводится для справки, объяснять его построчно мы не будем.)

Листинг 5.19. Метод MenuBuilder.addIntentOptions

```
public int addIntentOptions(int group, int id, int categoryOrder,
                            ComponentName caller,
                            Intent[] specifics,
                            Intent intent, int flags,
                            MenuItem[] outSpecificItems)
{
    PackageManager pm = mContext.getPackageManager();
    final List<ResolveInfo> lri =
        pm.queryIntentActivityOptions(caller, specifics, intent, 0);
    final int N = lri != null ? lri.size() : 0;

    if ((flags & FLAG_APPEND_TO_GROUP) == 0) {
```

```
      removeGroup(group);
   }

   for (int i=0; i<N; i++) {
      final ResolveInfo ri = lri.get(i);
      Intent rintent = new Intent(
         ri.specificIndex < 0 ? intent : specifics[ri.specificIndex]);
      rintent.setComponent(new ComponentName(
            ri.activityInfo.applicationInfo.packageName,
            ri.activityInfo.name));
      final MenuItem item = add(group, id, categoryOrder,
            ri.loadLabel(pm));
      item.setIntent(rintent);
      if (outSpecificItems != null && ri.specificIndex >= 0) {
         outSpecificItems[ri.specificIndex] = item;
      }
   }
   return N;
}
```

Одна из строк в листинге 5.19 выделена полужирным шрифтом; в ней создается элемент меню. Код перепоручает работу по созданию заголовка элемента меню классу ResolveInfo. Исходный код класса ResolveInfo показывает, что фильтр намерений, в котором было объявлено конкретное намерение, должен иметь отдельное название. Например:

```
<intent-filter android:label="Zagolovok Menu ">
   …….
   <category android:name="android.intent.category.ALTERNATE" />
   <data android:mimeType="dannye opredelennogo tipa" />
</intent-filter>
```

Значение label фильтра намерений служит названием меню. Чтобы посмотреть, как работает этот компонент, обратитесь к примеру с программой NotePad для Android.

Работа с меню при изменении данных

До сих пор мы говорили о статических меню — их параметры настраиваются при создании меню и не изменяются динамически в зависимости от того, что отображается на экране. Если вы хотите создать динамические меню, используйте метод onPrepareOptionsMenu, предоставляемый в Android. Он напоминает onCreateOptionsMenu, но в отличие от последнего вызывается при каждой активации меню. Метод onPrepareOptionsMenu используется, например, когда необходимо отключать определенные меню или группы меню в зависимости от того, какие данные отображаются на экране. Не забудьте об этом свойстве, проектируя набор функций меню.

И прежде, чем перейти к обсуждению диалоговых окон, мы должны обсудить еще один немаловажный аспект, связанный с меню. В Android при создании меню можно использовать XML-файлы. В следующем разделе мы рассмотрим поддержку XML-меню в Android.

Загрузка меню при помощи XML-файлов

До сих пор мы создавали меню средствами программирования. Такой способ создания меню — не самый удобный, так как для каждого меню вам приходится задавать несколько ID и определять константы для каждого из этих ID. Согласитесь, это достаточно скучная работа.

Вместо этого меню можно определять в XML-файлах. В Android это возможно, так как меню являются разновидностью ресурсов. При применении XML в ходе создания меню вы получаете ряд преимуществ, в частности возможность озаглавливать меню, автоматически их упорядочивать, определять для них ID и т. д. Обеспечивается также поддержка локализации текста меню.

При работе с XML-меню необходимо выполнить следующие операции.

1. Задать XML-файл с тегами меню.
2. Поместить файл в подкаталог /res/menu. Имя файла выбирается произвольно, в этом подкаталоге может быть любое количество файлов. Android автоматически генерирует ID для данного файла меню.
3. Применить ID ресурса для файла меню, чтобы загрузить в меню XML-файл.
4. Организовать отклик на элементы меню посредством ID ресурсов, сгенерированных для каждого элемента меню.

В следующих подразделах мы подробно поговорим о каждом из этих этапов и в качестве примера приведем соответствующие фрагменты кода.

Структура XML-файла ресурсов, относящегося к меню

Сначала рассмотрим XML-файл, в котором определяются меню (листинг 5.20). Все файлы меню начинаются с одного и того же тега верхнего уровня (menu), за которым следует ряд тегов group. Каждый из тегов group соответствует группе элементов меню — о таких группах мы говорили в начале главы. ID для группы можно указать методом @+id. В каждой группе меню будет содержаться ряд элементов, ID которых будут ассоциированы с символическими именами. В документации по Android SDK приводится список всех аргументов, которые могут быть у таких XML-тегов.

Листинг 5.20. XML-файл, в котором определяются меню

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <!--В этой группе используется категория, заданная по умолчанию. -->
   <group android:id="@+id/menuGroup_Main">

      <item android:id="@+id/menu_testPick"
         android:orderInCategory="5"
         android:title="Test Pick" />
      <item android:id="@+id/menu_testGetContent"
         android:orderInCategory="5"
```

```
        android:title="Test Get Content" />
    <item android:id="@+id/menu_clear"
        android:orderInCategory="10"
        android:title="clear" />
    <item android:id="@+id/menu_dial"
        android:orderInCategory="7"
        android:title="dial" />
    <item android:id="@+id/menu_test"
        android:orderInCategory="4"
        android:title="@+string/test" />
    <item android:id="@+id/menu_show_browser"
        android:orderInCategory="5"
        android:title="show browser" />
  </group>
</menu>
```

В XML-файле меню, показанном в листинге 5.20, присутствует одна группа. На основании определения ID @+id/menuGroup_main этой группе автоматически присваивается ID ресурса, называемый menuGroup_main in the R.java (файл ресурса ID). Аналогично для всех дочерних элементов меню выделяются ID элементов меню, создаваемые на базе символических определений ID, содержащихся в данном XML-файле.

Наполнение XML-файлов ресурсов, относящихся к меню

Допустим, у нас есть XML-файл, называемый my_menu.xml. Этот файл нужно будет поместить в подкаталоге /res/menu. Когда файл помещается в /res/menu, автоматически генерируется ID ресурса, называемый R.menu.my_menu.

Теперь посмотрим, как можно использовать такой ID ресурса меню при заполнении меню параметрами. В Android имеется класс android.view.MenuInflater, предназначенный для заполнения меню объектами, взятыми из XML-файлов. Мы используем экземпляр MenuInflater, чтобы задействовать ID ресурса R.menu.my_menu для наполнения меню данными:

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   MenuInflater inflater = getMenuInflater(); // от явления
   inflater.inflate(R.menu.my_menu, menu);

   // Чтобы можно было увидеть меню, должен быть возвращен результат true.
   return true;

}
```

В данном коде сначала получаем MenuInflater из класса Activity, а потом приказываем ему загрузить XML-файл непосредственно в меню.

Отклик на элементы меню, работающие на базе XML

Мы пока так и не рассмотрели особых преимуществ, появляющихся при использовании XML, — они становятся очевидны при отклике на элементы меню. Отклик на элементы XML-меню организуется почти так же, как и на элементы, создаваемые методами программирования, различия совсем невелики. Как и в предыдущем случае, при работе с элементами меню используется метод обратного вызова onOptionsItemSelected. Но на этот раз работа упрощается, благодаря некоторым свойствам ресурсов Android (ресурсы подробно рассмотрены в главе 3). Как было указано в подразделе «Структура XML-файла ресурсов, относящегося к меню», Android генерирует не только ID для XML-файла, но и необходимые ID для элементов меню, по которым эти элементы можно различать. Таким образом, отклик на элементы меню упрощается и усовершенствуется, так как вам не приходится специально создавать ID для элементов меню и управлять ими.

При более подробном рассмотрении данного вопроса следует отметить, что при использовании XML в ходе создания меню вам не приходится задавать константы для ID и беспокоиться об уникальности идентификаторов, так как эти задачи решаются при автоматическом генерировании ID. Данная функция проиллюстрирована в следующем коде:

```
private void onOptionsItemSelected (MenuItem item)
{
   this.appendMenuItemText(item);
   if (item.getItemId() == R.id.menu_clear)
   {
      this.emptyText();
   }
   else if (item.getItemId() == R.id.menu_dial)
   {
      this.dial();
   }
   else if (item.getItemId() == R.id.menu_testPick)
   {
      IntentsUtils.invokePick(this);
   }
   else if (item.getItemId() == R.id.menu_testGetContent)
   {
      IntentsUtils.invokeGetContent(this);
   }
   else if (item.getItemId() == R.id.menu_show_browser)
   {
      IntentsUtils.tryOneOfThese(this);
   }
}
```

Обратите внимание: названия элементов меню, которые берутся из XML-файла ресурсов, относящегося к меню, получают для элементов автоматически сгенерированные ID, относящиеся к диапазону R.id.

Краткое описание дополнительных XML-тегов, используемых при работе с меню

При создании XML-файлов вы должны знать различные XML-теги, которые будут использоваться при работе. Чтобы быстро получить справку по этому вопросу, посмотрите демоверсии API, имеющиеся в Android SDK. К таким демонстрационным API относятся наборы меню, на примере которых вы можете изучить любые аспекты программирования в Android. В подкаталоге /res/menu можно ознакомиться с примерами XML-меню. Ниже мы кратко рассмотрим некоторые наиболее важные теги.

Тег категории группы

При помощи тега menuCategory в XML-файле можно указать категорию группы:

```
<group android:id="@+id/some_group_id "
 android:menuCategory="secondary">
```

Теги контролируемой работы

Тег checkableBehavior используется для контроля отклика элементов на щелчки на них на уровне группы:

```
<group android:id="@+id/noncheckable_group"
   android:checkableBehavior="none">
```

Тег checked используется в тех же целях при работе с отдельно взятым элементом меню:

```
<item android:id=".."
 android:title="…"
 android:checked="true" />
```

Теги для моделирования работы подменю

Подменю представляет собой компонент, находящийся под элементом меню:

```
<item android:title="All without group">
 <menu>
         <item…>
 </menu>
</item>
```

Тег пиктограммы меню

Можно использовать тег icon, чтобы ассоциировать изображение с элементом меню:

```
<item android:id=".. "
 android:icon="@drawable/some-file" />
```

Тег для активации/деактивации меню

При помощи тега enabled элемент меню можно активировать или деактивировать:

```
<item android:id=".. "
 android:enabled="true"
 android:icon="@drawable/some-file" />
```

Комбинации для быстрого доступа к элементам меню

Чтобы создать комбинацию команд для быстрого доступа к элементу меню, используйте тег alphabeticShortcut:

```
<item android:id="… "
 android:alphabeticShortcut="a"
 …
</item>
```

Видимость меню

При помощи индикатора visible можно управлять видимостью и скрытостью элемента меню:

```
<item android:id="… "
   android:visible="true"
   …
</item>
```

Вот мы и рассмотрели меню параметров, подменю, пиктографические, контекстные и альтернативные меню. Кроме того, мы обсудили вопросы использования XML-меню и связанные с этим преимущества. Теперь обратимся к поддержке диалоговых окон в Android.

Использование диалоговых окон в Android

Если ранее вы занимались программированием в такой среде, в которой диалоговые окна являются синхронными (в частности, это касается модальных диалоговых окон), организация диалоговых окон в Android может показаться вам странной. В Android диалоговые окна являются асинхронными. При работе с модальными диалоговыми окнами такая асинхронность является несколько парадоксальной: это все равно, как если бы вы беседовали с кем-то, используя при этом только переднюю часть мозга, а при помощи задней части одновременно с этим решали совсем другую задачу. Однако модель split-brain, при которой вычислительная мощность разделяется для решения нескольких задач сразу, вполне неплоха, если речь идет о компьютерах. Такой асинхронный подход позволяет повысить скорость отклика мобильного устройства.

Диалоговые окна в Android являются не только асинхронными, но и *управляемыми* (managed); это означает, что диалоговые окна могут одновременно использоваться несколькими активирующими их процессами. Такая модель обусловлена

необходимостью оптимизации затрат памяти и мощности устройства при создании, отображении и свертывании диалоговых окон.

В следующих подразделах мы более подробно рассмотрим аспекты, касающиеся работы с диалоговыми окнами в Android. Мы изучим основные виды диалоговых окон, например окон с предупреждениями (alert dialog), покажем, как создавать и использовать такие окна. Затем научимся работать с диалоговыми окнами, содержащими *подсказки* (prompt dialogs), — через такие диалоговые окна пользователь получает приглашение ввести определенную информацию, и пользовательский ввод возвращается программе. Мы также покажем, как вы можете загружать в диалоговые окна собственные шаблоны видов.

После этого поговорим о том, как происходит управление диалоговыми окнами в Android, и исследуем протокол, на основании которого и с применением функций обратного вызова явления создаются диалоговые окна. Наконец, мы обратимся к используемому в Android протоколу управления диалоговыми окнами и представим его в обобщенном виде, чтобы наши асинхронные управляемые диалоговые окна работали как можно более гладко. Даже такое обобщенное представление может помочь вам при работе, но мы им не ограничимся, а объясним также, как именно организуется работа диалоговых окон внутри системы.

Создание диалоговых окон с предупреждениями

Начнем с диалоговых окон, в которых содержатся предупреждения. Обычно *предупреждение* (alert) — это простое сообщение, касающееся валидации форм или исправления ошибок. Рассмотрим следующий пример, связанный с исправлением ошибок, который часто встречается в HTML-документах:

```
if (validate(field1) == false)
{
   // Указать при помощи предупреждающего диалогового окна,
   // что введенное сообщение имеет неверный формат.
   showAlert("Информация, введенная в поле1, имеет неверный формат");
   // поместить поле в фокус
   // ..и продолжить
}
```

Такое диалоговое окно можно написать на JavaScript при помощи функции JavaScript `alert`, выводящей на экран обычное синхронное диалоговое окно, в котором есть только само сообщение и кнопка OK. После того как пользователь нажмет OK, выполнение программы продолжится. Это диалоговое окно является как модальным, так и синхронным, поскольку следующая строка кода не будет выполнена, пока функция `alert` не вернет результат.

Подобные диалоговые окна с предупреждениями удобны при исправлении ошибок. Но в Android диалоговые окна могут использоваться не только по такому прямому назначению. В системе также поддерживается построитель предупреждающих диалоговых окон (alert-dialog builder), инструмент общего назначения, используемый при создании диалоговых окон с предупреждениями и работе с ними.

Итак, вы сами можете создать предупреждающее диалоговое окно, применив класс `android.app.AlertDialog.Builder`. Этот класс-построитель предназначен для создания диалоговых окон и позволяет пользователям выполнять следующие операции:

- читать сообщение и отвечать на него «Да» или «Нет»;
- выбирать элемент из списка;
- выбирать из списка несколько элементов;
- просматривать ход работы программы;
- выбирать один из предлагаемых параметров;
- отвечать на подсказку перед тем, как продолжить работу с программой.

Мы покажем, как построить подобное диалоговое окно и активировать его из элемента меню. Такой подход, применяемый при работе с любыми диалоговыми окнами, подразделяется на следующие этапы.

1. Создание объекта `Builder`.
2. Установка параметров отображения: количества кнопок, списков элементов и т. д.
3. Определение для кнопок методов обратного вызова.
4. Задание объекту `Builder` команды построить диалоговое окно. Тип создаваемого диалогового окна определяется в зависимости от того, какие параметры имеет объект `Builder`.
5. Применение `dialog.show()` для отображения диалогового окна.

В листинге 5.21 показан код, в котором реализованы эти этапы.

Листинг 5.21. Построение и отображение диалогового окна с предупреждением

```
public class Alerts
{
   public static void showAlert(String message, Context ctx)
{
      // создание построителя
      AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
      builder.setTitle("Alert Window");

      // добавление кнопок и слушателя
      PromptListener pl = new EmptyListener();
      builder.setPositiveButton("OK", pl);

      // создание диалогового окна
      AlertDialog ad = builder.create();

      // отображение
      ad.show();
   }
}

public class EmptyListener
```

```
implements android.content.DialogInterface.OnClickListener {
   public void onClick(DialogInterface v, int buttonId)
   {
   }
}
```

Чтобы активировать код из листинга 5.21, можете создать в написанном ранее тестовом приложении новый элемент меню и организовать отклик при помощи следующего кода:

```
if (item.getItemId() == R.id.menu_simple_alert)
{
   Alerts.showAlert("Простое предупреждение", this);
}
```

Результат показан на рис. 5.4.

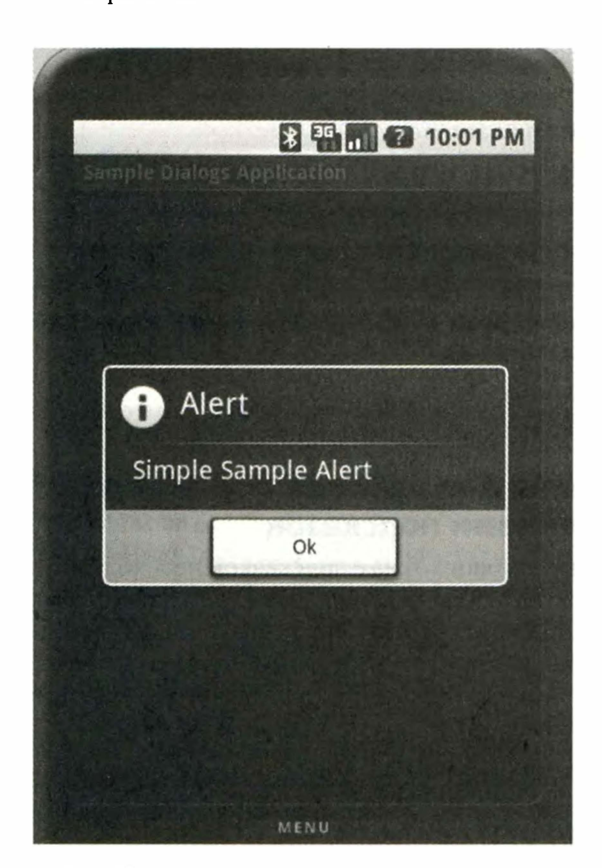


Рис. 5.4. Обычное диалоговое окно с предупреждением

Код для создания такого диалогового окна достаточно прост (см. листинг 5.21 и фрагмент кода после него). Код слушателя также очень прост. В принципе, после того, как нажата кнопка, мы ничего не делаем, просто создаем пустой слушатель,

который отслеживает щелчки на кнопке OK. Самая необычная деталь заключается в том, что вы не используете new при создании диалогового окна; вместо этого вы задаете параметры и приказываете построителю создать такое окно.

Создание диалоговых окон с подсказками

Итак, нам удалось создать простое диалоговое окно с предупреждением. Теперь сделаем другое окно, посложнее: диалоговое окно с подсказкой. Диалоговое окно с подсказкой — это еще один краеугольный камень JavaScript. В нем пользователь видит подсказку или вопрос, ответ на который вводится в текстовое поле редактирования (edit box). Окно возвращает полученную строку программе, которая после этого может продолжить работу. Такой пример будет очень полезно изучить, так как в нем используется много функций класса Builder и раскрывается синхронная, асинхронная, модальная и немодальная природа диалоговых окон в Android.

Ниже перечислены операции, которые необходимо выполнить, чтобы создать диалоговое окно с подсказкой.

1. Придумать вид-шаблон для диалогового окна с подсказкой.
2. Загрузить шаблон в класс View.
3. Создать объект Builder.
4. Установить вид в объекте Builder.
5. Задать кнопки и обратные вызовы для них, чтобы система могла работать с вводимым текстом.
6. Создать диалоговое окно, используя при этом построитель диалоговых окон с предупреждениями.
7. Отобразить диалоговое окно.

Ниже будет приведен код, используемый на каждом этапе.

XML-файл шаблона для работы с диалоговыми окнами подсказок

При отображении диалогового окна с подсказкой необходимо показать TextView, за которым будет следовать поле для ввода текста, куда пользователь может ввести ответ. В листинге 5.22 приведен XML-файл шаблона, предназначенный для создания диалогового окна с подсказкой. Чтобы вызывать файл prompt_layout.xml, его нужно сохранить в каталоге /res/layout, в результате чего будет создан ID ресурса R.layout.prompt_layout.

Листинг 5.22. Файл prompt_layout.xml File

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:orientation="vertical">

 <TextView
     android:id="@+id/promptmessage"
     android:layout_height="wrap_content"
```

```
    android:layout_width="wrap_content"
    android:layout_marginLeft="20dip"
    android:layout_marginRight="20dip"
    android:text="Это ваш текст"
    android:gravity="left"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<EditText
    android:id="@+id/editText_prompt"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:layout_marginLeft="20dip"
    android:layout_marginRight="20dip"
    android:scrollHorizontally="true"
    android:autoText="false"
    android:capitalize="none"
    android:gravity="fill_horizontal"
    android:textAppearance="?android:attr/textAppearanceMedium" />

</LinearLayout>
```

Настройка построителя предупреждающих диалоговых окон с пользовательским видом

При создании диалогового окна с подсказкой выполните этапы 2–4, упомянутые выше: загрузите XML-вид и задайте параметры построителя диалоговых окон с предупреждениями. В Android есть класс android.view.LayoutInflater, предназначенный для создания объекта View на основе XML-файла, в котором этот объект определен. Мы используем экземпляр LayoutInflater для заполнения вида, используемого в нашем диалоговом окне. При этом за основу возьмем XML-файл шаблона (листинг 5.23).

Листинг 5.23. Загрузка шаблона в диалоговое окно

```
LayoutInflater li = LayoutInflater.from(ctx);
View view = li.inflate(R.layout.promptdialog, null);

// получаем построитель и устанавливаем вид
AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
builder.setTitle("Prompt");
builder.setView(view);
```

В листинге 5.23 мы получили LayoutInflater при помощи метода LayoutInflater.from(ctx), а затем использовали объект LayoutInflater, чтобы заполнить создаваемый объект View информацией из файла XML. После этого мы указываем в качестве параметров построителя диалоговых окон с предупреждениями заголовок и только что созданный вид.

Создание кнопок и слушателей

Переходим к этапу 5: создаем кнопки. В нашем окне должны присутствовать кнопки OK и Cancel, при помощи которых пользователь сможет ответить на подсказку.

При нажатии Cancel программа не должна считывать какой-либо текст, введенный в окно с подсказкой. Если пользователь нажимает OK, программа получает значение из текстового поля и передает его обратно к явлению.

Чтобы создать эти кнопки, нужен слушатель, который будет отвечать на обратные вызовы. Код слушателя будет приведен в подразделе «Слушатель диалогового окна с подсказкой», а пока рассмотрим листинг 5.24, в котором представлен процесс создания кнопок.

Листинг 5.24. Создание кнопок OK и Cancel

```
// добавление кнопок и слушателя
PromptListener pl = new PromptListener(view,ctx);
builder.setPositiveButton("OK", pl);
builder.setNegativeButton("Cancel", pl);
```

В данном коде предполагается, что класс слушателя называется PromptListener. Мы зарегистрировали этот слушатель для отслеживания работы обеих кнопок.

Создание и отображение диалогового окна с подсказкой

Наконец, мы выполняем этапы 6 и 7: создаем диалоговое окно с подсказкой и показываем его. Имея построитель диалоговых окон с предупреждениями, сделать это совсем не сложно (листинг 5.25).

Листинг 5.25. Команда построителю диалоговых окон с предупреждениями создать диалоговое окно

```
// получение диалогового окна
AlertDialog ad = builder.create();
ad.show();

// возвращение подсказки
return pl.getPromptReply();
```

В последней строке кода используется слушатель, возвращающий ответ на подсказку. Теперь, как и обещали, покажем код класса PromptListener.

Слушатель диалогового окна с подсказкой

Диалоговое окно с подсказкой взаимодействует с явлением через класс обратного вызова слушателя, который называется PromptListener. В этом классе есть метод обратного вызова, называемый onClick. ID кнопки, передаваемый к onClick, указывает, кнопка какого типа была нажата. Оставшаяся часть кода вполне понятна (листинг 5.26). Когда пользователь вводит текст и нажимает OK, значение текста передается в поле promptReply. В противном случае сохраняется значение null.

Листинг 5.26. PromptListener, класс для обратного вызова слушателя

```
public class PromptListener
implements android.content.DialogInterface.OnClickListener
{
   // локальная переменная для возвращения значения ответа на подсказку
   private String promptReply = null;

   // сохранение переменной для вида с целью нахождения значения подсказки
```

```
    View promptDialogView = null;

    // включение вида в конструктор
    public PromptListener(View inDialogView) {
          promptDialogView = inDialogView;
    }

    // метод обратного вызова из диалоговых окон
    public void onClick(DialogInterface v, int buttonId) {
      if (buttonId == DialogInterface.BUTTON1) {
          // кнопка ok
          promptReply = getPromptText();
      }
      else {
          // кнопка cancel
          promptValue = null;
      }
  }

  // обычный метод доступа к информации, введенной в текстовое поле
  private String getPromptText() {
      EditText et = (EditText)
      promptDialogView.findViewById(R.id.promptEditTextControlId);
      return et.getText().toString();
  }
  public String getPromptReply() { return promptReply; }
}
```

Все вместе

Теперь, когда мы разобрали весь код, применяемый для создания диалогового окна с подсказкой, представим эти фрагменты вместе. Код из листинга 5.27 можно использовать, чтобы протестировать диалоговое окно. Мы исключили из кода класс PromptListener, так как он отдельно показан в листинге 5.26.

Листинг 5.27. Код для тестирования диалогового окна с подсказкой

```
public class Alerts
{
   public static String prompt(String message, Context ctx)
   {
      // загрузка какого-либо вида
      LayoutInflater li = LayoutInflater.from(ctx);
      View view = li.inflate(R.layout.promptdialog, null);

      // получение построителя и установка вида
      AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
      builder.setTitle("Prompt");
      builder.setView(view);

      // добавление кнопок и слушателя
      PromptListener pl = new PromptListener(view,ctx);
```

```
        builder.setPositiveButton("OK", pl);
        builder.setNegativeButton("Cancel", pl);

        // получение диалогового окна
        AlertDialog ad = builder.create();

        // показ
        ad.show();

        return pl.getPromptReply();
    }
}
```

Для активации кода из листинга 5.27 можно создать в тестовом приложении, описанном в начале этой главы, новый элемент меню, и организовать отклик на этот элемент меню при помощи следующего кода:

```
if (item.getItemId() == R.id.menu_simple_alert)
{
    String reply = Alerts.showPrompt("Your text goes here", this);
}
```

Результат показан на рис. 5.5.



Рис. 5.5. Простое диалоговое окно с подсказкой

Но, написав весь этот код, вы увидите, что диалоговое окно с подсказкой всегда возвращает null, даже если пользователь введет текст в поле. Оказывается, в следующем коде метод show() активирует диалоговое окно асинхронным образом:

```
ad.show() //dialog.show
return pl.getPromptReply(); // listener.getpromptReply()
```

Это означает, что метод getPromptReply() вызывается в ответ на значение подсказки до того, как пользователь успеет ввести текст и нажать кнопку OK. Чтобы понять, почему происходит эта ошибка, нужно досконально изучить сущность диалоговых окон в Android.

Сущность диалоговых окон в Android

Как мы уже говорили, отображение диалоговых окон в Android — это асинхронный процесс. Как только диалоговое окно появится на экране, основной поток, активировавший это окно, возвращается к более раннему этапу задачи и обрабатывает оставшийся код. Это не означает, что диалоговое окно не является *модальным*. Оно именно модальное. Щелчки кнопкой мыши применяются только к диалоговому окну, а родительское явление возвращается к обработке своего цикла сообщений.

В некоторых оконных системах модальные диалоговые окна работают немного иначе. Вызывающий оператор блокируется, и пользователь вводит ответ в диалоговое окно. (Блок может быть виртуальным, а не реальным.) В операционной системе Windows поток, обеспечивающий отправку сообщений, начинает отправлять информацию в диалоговое окно и приостанавливает отправку данных в родительское окно (parent window). После закрытия диалогового окна поток возвращается к работе с родительским окном. Такой вызов является синхронным.

Подобный подход неприменим при работе с мобильным устройством, где чаще происходят неожиданные события, и реагировать на них должен именно основной поток. Для обеспечения надежного отклика системы на этом уровне Android сразу же возвращает основной поток к обработке цикла сообщений.

При реализации такой модели подразумевается, что в системе отсутствуют простые диалоговые окна, в которых запрашивается ответ, и работа системы останавливается до тех пор, пока ответ не придет. Модель программирования должна содержать иной принцип интеграции обратных вызовов.

Переработка диалогового окна с подсказкой

Пересмотрим код, с которым у нас возникли проблемы при написании диалогового окна с подсказкой:

```
if (item.getItemId() == R.id.menu_simple_alert)
{
   String reply = Alerts.showPrompt("Your text goes here", this);
}
```

Как показывает практика, строковая переменная reply будет иметь значение null, так как диалоговое окно с подсказкой, активированное при помощи Alerts.showPrompt(), не может вернуть значение тому же потоку. Единственный способ

решения этой задачи — приказать явлению реализовать метод обратного вызова непосредственно, а не при помощи класса PromptListener. Эта операция делается в классе Activity путем реализации OnClickListener:

```
public class SampleActivity extends Activity
implements android.content.DialogInterface.OnClickListener
{
...... prochij kod

if (item.getItemId() == R.id.menu_simple_alert)
{
   Alerts.showPrompt("Your text goes here", this);
}
.....
public void onClick(DialogInterface v, int buttonId)
{
   // определение способа считывания строки из диалогового окна
   // и ответа на нее
}
```

Из этого метода обратного вызова onClick видно, что вы можете правильно считывать переменные из инстанцированного диалогового окна, так как пользователь уже закроет диалоговое окно к тому моменту, как будет вызван метод.

Применение диалоговых окон, таким образом, является совершенно правомерным. Но в Android есть и дополнительный механизм, улучшающий производительность системы. Речь идет об *управляемых диалоговых окнах*, такие диалоговые окна используются одновременно несколькими активными процессами. При работе с управляемыми диалоговыми окнами вам по-прежнему не обойтись без обратных вызовов. На самом деле все навыки, приобретенные вами при разработке диалоговых окон с подсказками, пригодятся и при работе с управляемыми диалоговыми окнами — вы поймете как принципы их работы, так и тонкости их создания.

Работа с управляемыми диалоговыми окнами

В Android используется специальный протокол для работы с управляемыми диалоговыми окнами, позволяющий повторно использовать созданные ранее экземпляры (instances) диалоговых окон, а не создавать новые диалоговые окна в ответ на действия. Однако мы считаем, что этот протокол делает работу с диалоговыми окнами скучной и рутинной. Мы изучим его, а потом разработаем небольшой фреймворк, в котором обобщим основные черты этого протокола и поможем вам упростить работу с управляемыми диалоговыми окнами.

Протокол управляемых диалоговых окон

Основная цель протокола управляемых диалоговых окон заключается в повторном использовании диалогового окна, которое уже активировалось ранее при данном сеансе работы с программой. Эта техника напоминает использование

объектного пула в Java. Протокол управляемых диалоговых окон работает по следующей схеме.

1. Присвоение уникального ID каждому диалоговому окну, которое вы хотите создать и использовать. Предположим, одно из диалоговых окон имеет отметку 1.
2. Команда Android отобразить диалоговое окно номер 1.
3. Android проверяет, есть ли уже в выполняемом явлении диалоговое окно с номером 1. Если такое диалоговое окно имеется, Android снова отображает его, не пересоздавая. Перед показом диалогового окна Android вызывает функцию `onPrepareDialog()`, очищающую диалоговое окно от неактуальной информации.
4. Если нужное диалоговое окно еще не существует, Android вызывает метод `onCreateDialog`, сообщая ему ID диалогового окна (в данном случае — 1).
5. Программист (то есть вы) переопределяет метод `onCreateDialog`. Необходимо создать диалоговое окно, воспользовавшись построителем диалоговых окон с предупреждениями, который вернет соответствующее значение. Однако перед созданием диалогового окна код должен определить, диалоговое окно с каким ID необходимо создать. Для решения этой задачи вам потребуется оператор `switch`.
6. Android отображает диалоговое окно.
7. При нажатии кнопок диалоговое окно активирует обратные вызовы.

Теперь воспользуемся этим протоколом и переделаем наше неуправляемое диалоговое окно с предупреждением в управляемое.

Преобразование неуправляемого диалогового окна в управляемое

Мы выполним все описанные выше шаги и переделаем диалоговое окно с предупреждением. Сначала зададим для этого окна уникальный ID в контексте указанного явления:

```
// уникальный id диалогового окна
private static final int DIALOG_ALERT_ID = 1;
```

Это достаточно просто. Мы создали ID, соответствующий диалоговому окну и упрощающий управление обратными вызовами. Такой ID позволяет произвести с элементом меню следующую операцию:

```
if (item.getItemId() == R.id.menu_simple_alert)
{
   showDialog(this.DIALOG_ALERT_ID);
}
```

Метод `showDialog`, входящий в состав Android SDK, инициирует вызов метода `onCreateDialog()`. Android достаточно интеллектуален и не вызывает `onCreateDialog()` многократно. При вызове этого метода нам нужно создать диалоговое окно и вернуть его Android. Внутри системы Android сохранит это окно, и его можно будет

использовать повторно. Ниже приведен пример кода для создания диалогового окна на базе уникального ID:

```
@Override
protected Dialog onCreateDialog(int id) {
   switch (id) {
      case DIALOG_ALERT_ID:
         return createAlertDialog();
   }
   return null;
}

private Dialog createAlertDialog()
{
   AlertDialog.Builder builder = new AlertDialog.Builder(this);
   builder.setTitle("Предупреждение");
   builder.setMessage("сообщение");
   EmptyOnClickListener emptyListener = new EmptyOnClickListener();
   builder.setPositiveButton("Ok", emptyListener );
   AlertDialog ad = builder.create();
   return ad;
}
```

Обратите внимание, как метод onCreateDialog() должен узнавать входящий ID и находить соответствующее ему диалоговое окно. Сам createAlertDialog() содержится в отдельной функции и параллельно выполняет процессы создания диалоговых окон, рассмотренных в предыдущих разделах. В коде также используется тот самый EmptyOnClickListener, который применялся при работе с предупреждающими диалоговыми окнами.

Поскольку диалоговое окно создается только один раз, нам нужен специальный механизм, чтобы изменять определенные параметры диалогового окна при каждом показе. Таким механизмом является метод обратного вызова onPrepareDialog():

```
@Override
protected void onPrepareDialog(int id, Dialog dialog) {
   switch (id) {
   case DIALOG_ALERT_ID:
      prepareAlertDialog(dialog);
   }
}

private void prepareAlertDialog(Dialog d) {
   AlertDialog ad = (AlertDialog)d;
   // изменить что-нибудь в этом диалоговом окне
}
```

При наличии такого кода showDialog(1) будет работать. Даже если бы вы вызывали этот метод многократно, метод onCreateMethod вызывался бы только один раз. При помощи этого протокола вы также можете переделать диалоговое окно с подсказкой.

Итак, организация отклика на обратные вызовы диалоговых окон — это кусок работы, но при применении протокола управляемых диалоговых окон работы становится еще больше. После того как авторы книги попользовались протоколом

управляемых диалоговых окон, возникла мысль обобщить идеи, реализованные в этом протоколе, и перестроить его так, чтобы решались две задачи:

- вынесение операций идентификации и создания диалогов за пределы класса явления;
- выполнение операций идентификации и создания диалогов в специальном классе для диалоговых окон.

В следующем подразделе мы подробно изучим разработку такого фреймворка, а затем с его помощью переделаем предупреждающее диалоговое окно и диалоговое окно с подсказкой.

Упрощение протокола управляемых диалоговых окон

Как видите, работа с управляемыми предупреждающими диалоговыми окнами может получиться достаточно запутанной, основной код быстро наполнится посторонней информацией. Если вычленить из этого протокола важнейшую информацию и упростить его, он будет выглядеть так.

1. Создание экземпляра нужного нам диалогового окна (например, с названием `dialog1`) путем применения `new` и сохранения его в локальной переменной.
2. Вывод диалогового окна при помощи `dialog1.show()`.
3. Реализация метода в явлении, называемом `dialogFinished()`.
4. Считывание при помощи метода `dialogFinished()` атрибутов `dialog1`, например `dialog1.getValue1()`.

При использовании такого алгоритма отображение управляемого предупреждающего диалогового окна будет происходить так:

```
….class MyActivity ….
{
   // новое диалоговое окно
   ManagedAlertDialog mad = new ManagedAlertDialog("message", …, .. );

   ….opredelennyj metod menu ,
   if (item.getItemId() == R.id.menu_simple_alert)
   {
      // показ диалогового окна
      mad.show();
   }
   …..
   // Доступ к внутренней структуре этого глупого диалогового окна —
   // если хотите, конечно.
   dialogFinsihed()
   {
      ….
      // использование значений диалогового окна
      mad.getA();
      mad.getB();
   }
}
```

Позволим себе предположить, что такой метод работы с диалоговыми окнами гораздо проще канонического. Отпадает необходимость запоминать ID, не нужно засорять основной код командами, предназначенными для создания диалогового окна, а объекты, производные от диалоговых окон, можно использовать непосредственно для доступа к значениям.

Принцип такой абстракции описан ниже. Сначала мы обобщаем операции создания диалогового окна и его подготовки в отдельном классе, идентифицирующем базовое диалоговое окно. Этот интерфейс мы называем `IDialogProtocol`. Метод `show()` применяется непосредственно к этому диалоговому окну. Диалоговые окна собираются в группу и сохраняются в реестре в базовом классе явления, их ID используются как ключи. Базовое явление демультиплексирует вызовы `onCreate`, `onPrepare` и `onClick` на основе их ID и переадресует их к классу диалогового окна. Архитектура данного процесса более подробно описана на рис. 5.6.

В листинге 5.28 показано, как используется такой фреймворк.

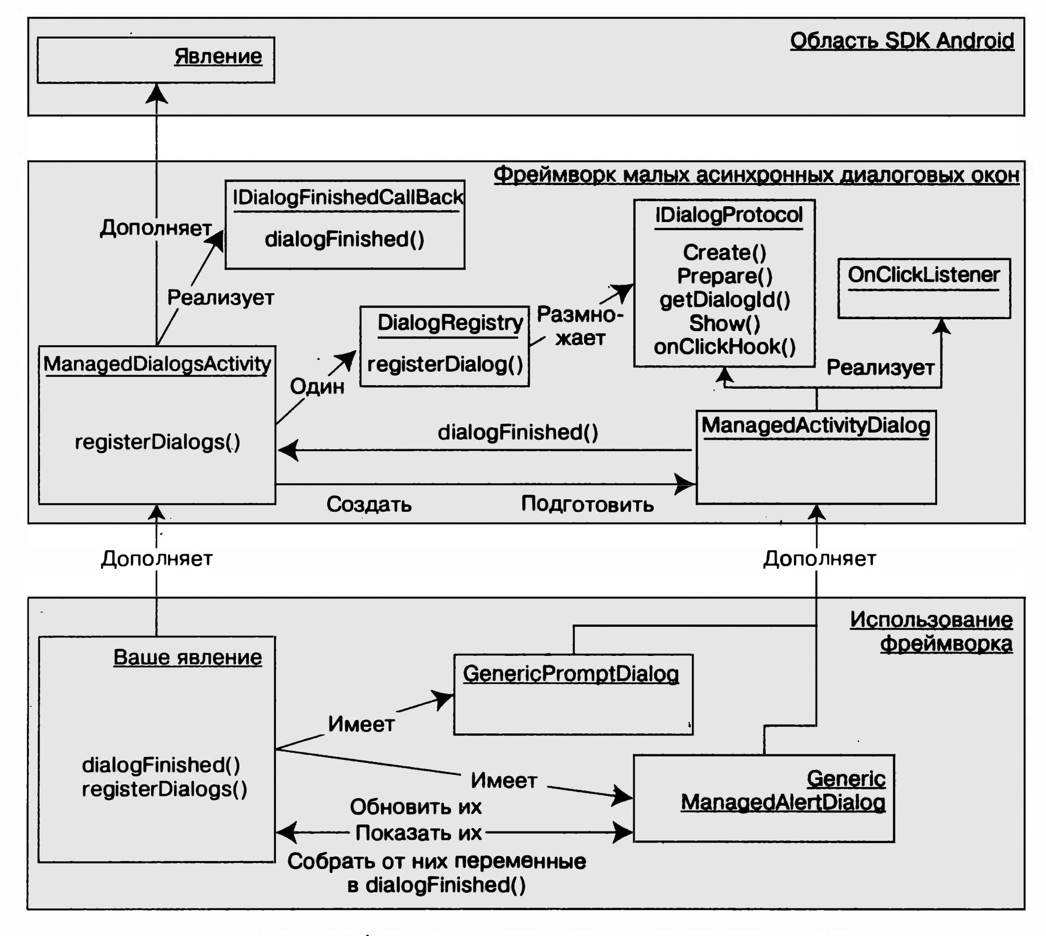


Рис. 5.6. Простой фреймворк для управления диалоговыми окнами

Листинг 5.28. Сокращенный вариант протокола управления диалоговыми окнами

```
public class MainActivity extends ManagedDialogsActivity
{
   // диалоговое окно 1
   private GenericManagedAlertDialog gmad =
      new GenericManagedAlertDialog(this,1,"InitialValue");

   // диалоговое окно 2
   private GenericPromptDialog gmpd =
      new GenericPromptDialog(this,2,"InitialValue");

   // элементы меню для начала работы с диалоговым окном
   else if (item.getItemId() == R.id.menu_simple_alert)
   {
      gmad.show();
   }
   else if (item.getItemId() == R.id.menu_simple_prompt)
   {
      gmpd.show();
   }

   // работа с обратными вызовами
   public void dialogFinished(ManagedActivityDialog dialog, int buttonId)
   {
      if (dialog.getDialogId() == gmpd.getDialogId())
      {
         String replyString = gmpd.getReplyString();
      }
   }
}
```

Чтобы задействовать этот фреймворк, сначала дополним ManagedDialogsActivity. Потом инстанцируем нужные нам диалоговые окна — все они будут производными от ManagedActivityDialog. Для организации отклика на элементы меню можно просто применить метод show(). Диалоговые окна сами принимают необходимые параметры, и только после этого такие окна отображаются. Хотя мы передаем ID диалогового окна, помнить такой ID нам уже не нужно. Можете вообще не думать об этих ID.

Теперь отдельно рассмотрим каждый из классов, показанных на рис. 5.6.

IDialogProtocol

Интерфейс IDialogProtocol определяет, каким будет управляемое диалоговое окно. Управляемое диалоговое окно должно автоматически создаваться и отображаться всякий раз, когда это потребуется. Целесообразно передать эту функцию show самому диалоговому окну. Диалоговое окно должно само распознавать нажатия кнопок и вызывать соответствующий родительский объект, отвечающий

за закрытие окна. В следующем коде интерфейса эти идеи представлены в виде функций:

```
public interface IDialogProtocol
{
 public Dialog create();
 public void prepare(Dialog dialog);
 public int getDialogId();
 public void show();
 public void onClickHook(int buttonId);
}
```

ManagedActivityDialog

Абстрактный класс ManagedActivityDialog обеспечивает стандартную реализацию всех классов диалоговых окон, в которых предполагается использовать интерфейс IDialogProtocol. При этом функции create и prepare переопределяются базовыми классами и одновременно реализуются оставшиеся методы IDialogProtocol. Класс ManagedActivityDialog также сообщает родительскому явлению о том, что диалоговое окно закончило работу после отклика на нажатие кнопки. При этом используется схема меток в шаблонах (template-hook pattern), позволяющая производным классам конкретизировать метод перехвата ответа onClickHook. В этом же классе осуществляется переадресация метода show() к родительскому явлению. Таким образом, реализация show() становится более логичной. Класс ManagedActivityDialog следует использовать в качестве базового при создании всех новых диалоговых окон (листинг 5.29).

Листинг 5.29. Класс ManagedActivityDialog

```
public abstract class ManagedActivityDialog implements IDialogProtocol
 ,android.content.DialogInterface.OnClickListener
{
 private ManagedDialogsActivity mActivity;
 private int mDialogId;
 public ManagedActivityDialog(ManagedDialogsActivity a, int dialogId)
 {
     mActivity = a;
     mDialogId = dialogId;
 }
 public int getDialogId()
 {
     return mDialogId;
 }
 public void show()
 {
     mActivity.showDialog(mDialogId);
 }
 public void onClick(DialogInterface v, int buttonId)
 {
     onClickHook(buttonId);
     this.mActivity.dialogFinished(this, buttonId);
 }
}
```

DialogRegistry

Класс DialogRegistry позволяет решить две задачи. В нем соблюдается соответствие между ID диалоговых окон и экземплярами конкретных диалоговых окон (или фабрик). Он также передает общие вызовы onCreate и onPrepare определенным диалоговым окнам путем сопоставления ID с объектами (D-to-object). ManagedDialogsActivity использует класс DialogRegistry как репозиторий для регистрации новых диалоговых окон (листинг 5.30).

Листинг 5.30. Класс DialogRegistry

```
public class DialogRegistry
{
   SparseArray<IDialogProtocol> idsToDialogs
                              = new SparseArray();
   public void registerDialog(IDialogProtocol dialog)
   {
      idsToDialogs.put(dialog.getDialogId(),dialog);
   }

   public Dialog create(int id)
   {
      IDialogProtocol dp = idsToDialogs.get(id);
      if (dp == null) return null;

      return dp.create();
   }
   public void prepare(Dialog dialog, int id)
   {
      IDialogProtocol dp = idsToDialogs.get(id);
      if (dp == null)
      {
         throw new RuntimeException("Dialog id is not registered:" + id);
      }
      dp.prepare(dialog);
 }
}
```

ManagedDialogsActivity

Класс ManagedDialogsActivity является базовым классом явлений, в которых поддерживаются управляемые диалоговые окна. В нем сохраняется одиночный экземпляр DialogRegistry, используемый для отслеживания управляемых диалоговых окон, идентифицируемых в интерфейсе IDialogProtocol. Класс позволяет производным явлениям регистрировать свои диалоговые окна при помощи функции registerDialogs(). На рис. 5.6 показано, что этот класс также отвечает за передачу семантики create и prepare соответствующему экземпляру диалогового окна, а кроме того, за нахождение этого диалогового окна в реестре. Наконец, в этом классе содержится метод dialogFinished для работы со всеми диалоговыми окнами, находящимися в реестре (листинг 5.31).

Листинг 5.31. Класс ManagedDialogsActivity

```
public class ManagedDialogsActivity extends Activity
        implements IDialogFinishedCallBack
{
 // реестр для управляемых диалоговых окон
 private DialogRegistry dr = new DialogRegistry();

 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     this.registerDialogs();
 }

 protected void registerDialogs()
 {
     // Ничего не делать,
     // если производные классы переопределят этот метод
     // для регистрации своих диалоговых окон.
     // Пример:
     // registerDialog(this.DIALOG_ALERT_ID_3, gmad);

 }
 public void registerDialog(IDialogProtocol dialog)
 {
     this.dr.registerDialog(dialog);
 }

 @Override
 protected Dialog onCreateDialog(int id) {
         return this.dr.create(id);
 }
 @Override
 protected void onPrepareDialog(int id, Dialog dialog) {
         this.dr.prepare(dialog, id);
 }

 public void dialogFinished(ManagedActivityDialog dialog, int buttonId)
 {
     // Ничего не делать,
     // если производные классы переопределят этот код.
 }
}
```

IDialogFinishedCallBack

Интерфейс IDialogFinishedCallBack позволяет классу ManagedActivityDialog сообщить родительскому явлению, что диалоговое окно закончило работу и что родительское явление может вызвать методы, предназначенные для получения параметров от диалогового окна. Обычно этот интерфейс реализуется при помощи ManagedDialogsActivity, которое выступает в качестве родительского явления ManagedActivityDialog (листинг 5.32).

Листинг 5.32. Интерфейс IDialogFinishedCallBack

```
public interface IDialogFinishedCallBack
{
 public static int OK_BUTTON = -1;
 public static int CANCEL_BUTTON = -2;
 public void dialogFinished(ManagedActivityDialog dialog, int buttonId);
}
```

GenericManagedAlertDialog

GenericManagedAlertDialog — это реализация диалогового окна. Этот класс дополняет ManagedActivityDialog. Данный класс отвечает за создание конкретного предупреждающего окна с применением построителя таких окон. Он также содержит информацию, которая будет использоваться в качестве локальных переменных. Поскольку GenericManagedAlertDialog реализует обычное предупреждающее диалоговое окно, он не использует метод onClickHook. Здесь важно подчеркнуть, что при использовании такого подхода в GenericManagedAlertDialog упакована сразу вся информация, относящаяся к процессу (листинг 5.33). При применении такого подхода основной код явления остается чистым, если не сказать — стерильным.

Листинг 5.33. Класс GenericManagedAlertDialog

```
public class GenericManagedAlertDialog extends ManagedActivityDialog
{
 private String alertMessage = null;
 private Context ctx = null;
 public GenericManagedAlertDialog(ManagedDialogsActivity inActivity,
                      int dialogId,
                      String initialMessage)
 {
     super(inActivity,dialogId);
     alertMessage = initialMessage;
     ctx = inActivity;
 }
 public Dialog create()
 {
     AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
     builder.setTitle("Alert");
     builder.setMessage(alertMessage);
     builder.setPositiveButton("Ok", this );
     AlertDialog ad = builder.create();
     return ad;
 }

 public void prepare(Dialog dialog)
 {
     AlertDialog ad = (AlertDialog)dialog;
     ad.setMessage(alertMessage);
 }
 public void setAlertMessage(String inAlertMessage)
 {
```

```
        alertMessage = inAlertMessage;
    }
    public void onClickHook(int buttonId)
    {
        // ничего не делать
        // не устанавливать локальных переменных
    }
}
```

GenericPromptDialog

В классе GenericPromptDialog заключена вся информация, необходимая для создания диалогового окна с подсказкой. Этот класс дополняет ManagedActivityDialog и предоставляет необходимые методы create и prepare (листинг 5.34). Вы также могли заметить, что здесь текст ответа сохраняется в локальной переменной, так, что родительское явление может получить к нему доступ при помощи метода обратного вызова dialogFinished.

Листинг 5.34. Класс GenericPromptDialog

```
public class GenericPromptDialog extends ManagedActivityDialog
{
   private String mPromptMessage = null;
   private View promptView = null;
   String promptValue = null;
   private Context ctx = null;
   public GenericPromptDialog(ManagedDialogsActivity inActivity,
         int dialogId,
         String promptMessage)
   {
      super(inActivity,dialogId);
      mPromptMessage = promptMessage;
      ctx = inActivity;
   }
   public Dialog create()
   {
      LayoutInflater li = LayoutInflater.from(ctx);
      promptView = li.inflate(R.layout.promptdialog, null);
      AlertDialog.Builder builder = new AlertDialog.Builder(ctx);
      builder.setTitle("prompt");
      builder.setView(promptView);
      builder.setPositiveButton("OK", this);
      builder.setNegativeButton("Cancel", this);
      AlertDialog ad = builder.create();
      return ad;
   }
   public void prepare(Dialog dialog)
   {
      // пока ничего
   }
   public void onClickHook(int buttonId)
   {
```

```
        if (buttonId == DialogInterface.BUTTON1)
        {
            // кнопка ok
            String promptValue = getEnteredText();
        }
    }
    private String getEnteredText()
    {
        EditText et =
            (EditText)
            promptView.findViewById(R.id.editText_prompt);
        String enteredText = et.getText().toString();
        Log.d("xx",enteredText);
        return enteredText;
    }
}
```

Резюме

В этой главе мы подробно рассмотрели меню и диалоговые окна, используемые в Android, — основные элементы, применяемые при программировании пользовательских интерфейсов. Вы научились работать с различными видами меню Android. Кроме того, вы узнали, как оптимизировать работу с меню при помощи XML-ресурсов.

Вы создали приложение для тестирования меню, которое, разумеется, можно использовать и при тестировании готовых программ. Через меню можно легко запускать и тестировать новые функции.

Вы также изучили диалоговые окна, с которыми в Android связаны специфические проблемы. Мы рассмотрели характерные черты асинхронных диалоговых окон и показали абстракцию, позволяющую упростить работу с управляемыми диалоговыми окнами.

Знания, приобретенные вами в этой главе, должны стать хорошей базой при написании сложных программ с пользовательскими интерфейсами. Кроме того, эти знания пригодятся вам при чтении следующей главы, в которой речь пойдет об анимации.

6 2D-анимация: премьера

В предыдущих главах мы подробно рассмотрели вопросы программирования пользовательских интерфейсов в Android. В этой главе мы научим вас создавать еще более понятные и привлекательные приложения для платформы Android — для этого мы изучим средства анимации, входящие в состав Android SDK. Наш опыт послужит вам лишь руководством, так как анимация требует творческого подхода от самого программиста.

В процессе анимации изменяется цветовая гамма, положение, размер или ориентация объекта на экране. В Android поддерживается три типа анимации: *покадровая анимация* (frame-by-frame animation), при использовании которой кадры одной серии последовательно отрисовываются через регулярные временные интервалы; *анимация шаблонов* (layout animation), при которой анимируется вид внутри контейнера, например список или таблица, и *анимация видов* (view animation), при которой анимируется любой обычный вид. Два последних типа относятся к так называемой *анимации с построением промежуточных кадров* (tweening animation), в которой между основными рисунками присутствуют вспомогательные. Идея заключается в том, что, если известно начальное и конечное состояние рисунка, художник может изменять определенные характеристики рисунка с течением времени. К таким характеристикам относятся цвет, положение, размер и т. д. На компьютере подобная анимация реализуется путем изменения средних значений через регулярные промежутки времени и перерисовывания поверхности. В этой главе мы изучим все упомянутые типы анимации, покажем рабочие примеры и проведем глубокий анализ.

Покадровая анимация — это наиболее простой из перечисленных типов. Мы займемся ею в первом разделе главы, рассмотрим, как она работает, как с ее помощью рассказать историю и как применить класс `AnimationDrawable`, чтобы кадры обновлялись с заданной частотой. Мы приведем пример со скриншотами и кодом, в котором создадим анимированное изображение шарика, движущегося по окружности.

Во втором разделе рассмотрим анимацию шаблонов, которая более сложна, чем покадровая, но проще, чем анимация видов. Поговорим о *масштабируемой анимации* (scale animation), при которой изменяется размер элемента; *транслируемой анимации* (translate animation), при которой элемент перемещается; *анимации вращения* (rotate animation), при которой изменяется ориентация элемента и *альфа-анимации* (alpha animation), при которой изменяется градиент цвета. Мы покажем,

как объявлять эти виды анимации в XML-файле и ассоциировать ID анимационных объектов с контейнерными видами, например со списками (list box). В качестве примеров опробуем анимационные трансформации с сериями текстовых элементов в списке. Кроме того, рассмотрим интерполяторы, которые определяют частоту смены кадров анимации, и анимационные наборы, в которых будут содержаться коллекции отдельных рисунков.

Последний раздел главы будет посвящен анимации видов. При таком типе анимации используются матрицы преобразований (transformation matrices). Чтобы усвоить материал этого раздела, вам будет нужно хорошо разобраться в матрицах преобразований, поэтому мы покажем несколько примеров, в которых используются такие матрицы. Кроме того, в Android имеется так называемая Camera, которая позволяет просматривать трехмерные изображения, проецируя 2D-вид, перемещающийся в 3D-пространстве. В этом разделе мы проиллюстрируем эти идеи, взяв ListView и повернув его в трехмерном пространстве.

Покадровая анимация

Покадровая анимация — это простой процесс, при котором с короткими интервалами отображаются серии идущих друг за другом изображений. В итоге получается эффект движения объекта. Именно по такому принципу работают кинопроекторы. Мы рассмотрим пример, в котором создадим изображение и сохраним его в виде набора картинок, незначительно отличающихся друг от друга. Затем мы возьмем набор таких изображений и при помощи кода запрограммируем их последовательный показ, имитировав таким образом анимацию.

Планирование покадровой анимации

Прежде чем приступить к написанию кода, нужно спланировать анимационную последовательность, в которой будет использоваться серия картинок. Пример изображения показан на рис. 6.1. На нем мы видим равные окружности, в различных точках которых расположен цветной шарик. Можно взять серию картинок, на которых круг находится в одном и том же месте, а шарик будет располагаться в разных точках окружности. Сохранив семь-восемь таких кадров, вы сможете применить анимацию, чтобы шарик как будто двигался по окружности.

Назовем наше изображение colored-ball. Сохраните восемь таких изображений в подкаталоге /res/drawable, чтобы потом к ним можно было получить доступ по их ID ресурсов. Название каждого изображения будет иметь вид colored-ballN, где N — это цифра, обозначающая номер изображения. Готовый проект с анимацией должен будет выглядеть так, как на рис. 6.2.

Большую часть этого явления занимает анимированный вид. Кроме того, здесь есть кнопка, позволяющая запускать и останавливать анимацию. Так мы сможем протестировать работу нашего приложения. В верхней части экрана мы поместили окно для записи диагностических сообщений, в котором вы сможете записывать все важные события, происходящие по ходу тестирования программы. Рассмотрим, как создать шаблон для такого явления.

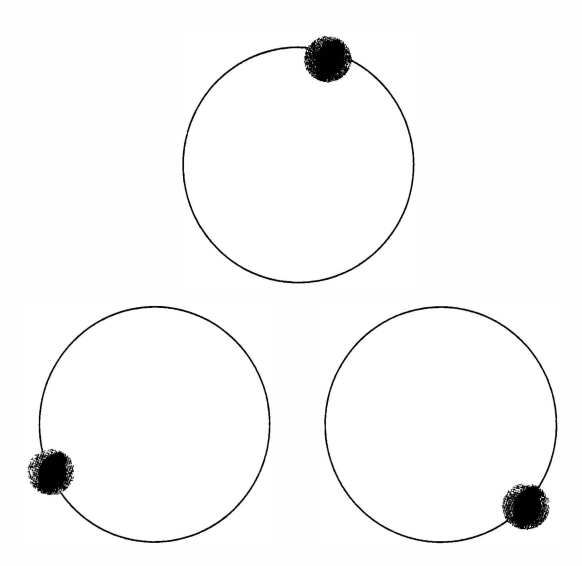


Рис. 6.1. Планирование анимации до написания кода

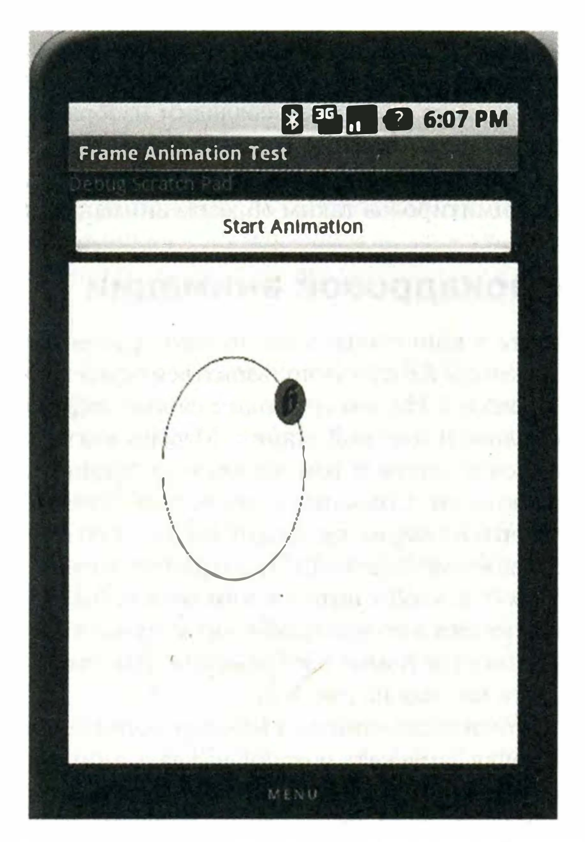


Рис. 6.2. Приложение для тестирования покадровой анимации

Создание явления

Начнем с создания базового файла шаблона, который будет написан на XML, и сохраним этот файл в подкаталоге /res/layout (листинг 6.1).

Листинг 6.1. XML-файл шаблона для создания примера с покадровой анимацией

```
<?xml version="1.0" encoding="utf-8"?>
<!--имя файла: /res/layout/frame_animations_layout.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView android:id="@+id/textViewId1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Debug Scratch Pad"
   />
<Button
   android:id="@+id/startFAButtonId"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Start Animation"
/>
<ImageView
   android:id="@+id/animationImage"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   />
</LinearLayout>
```

Первый элемент управления — это текстовое поле для записи диагностических сообщений, представляющее собой простой TextView. Затем мы добавим кнопку для запуска и остановки анимации. Последний вид — это ImageView, в котором будет отображаться анимация. Когда шаблон будет готов, создадим явление для загрузки этого вида (листинг 6.2).

Листинг 6.2. Явление для загрузки ImageView

```
public class FrameAnimationActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.frame_animations_layout);
   }
}
```

Это явление вы сможете загрузить из любого элемента меню того приложения, с которым сейчас работаете. Для этого понадобится выполнить следующий код:

```
Intent intent = new Intent(inActivity,FrameAnimationActivity.class);
inActivity.startActivity(intent);
```

Теперь явление будет выглядеть так, как на рис. 6.3.

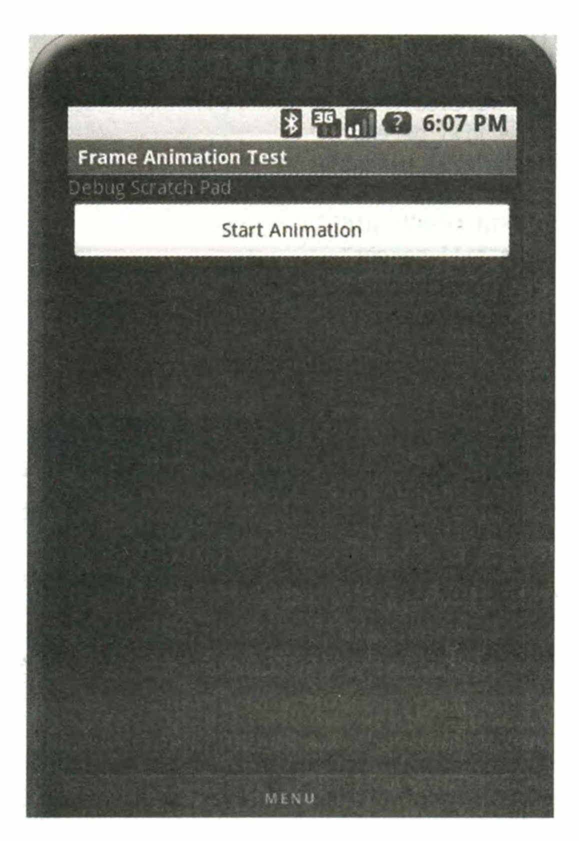


Рис. 6.3. Явление для покадровой анимации

Анимирование явления

Теперь у нас есть готовое явление и шаблон для него. Добавим в этот проект анимацию. В Android за покадровую анимацию отвечает специальный класс, имеющийся в графическом пакете. Он называется AnimationDrawable. Из названия класса понятно, что объекты из этого класса, как и любые другие отрисовываемые объекты, можно использовать как фон для любого вида. Например, фоновые точечные рисунки будут представлены как Drawables. Класс AnimationDrawable не только является Drawable, но и может содержать список других Drawable-ресурсов (например, изображений) и показывать их с заданным интервалом. Этот класс, по существу, является тонкой оболочкой (wrapper) для анимации, предоставляемой в базовом классе Drawable.

СОВЕТ

В классе Drawable для запуска анимации контейнер или вид получает команду активировать класс Runnable, который, по существу, перерисовывает Drawable с применением ряда параметров. Обратите внимание — для работы с классом AnimationDrawable не требуется знать подробности о внутренней реализации анимации. Но если перед вами стоит сравнительно сложная задача, можете посмотреть исходный код AnimationDrawable, который поможет вам написать собственные анимационные протоколы.

Чтобы задействовать класс AnimationDrawable, начните с набора ресурсов Drawable, которые находятся в подкаталоге /res/drawable. Например, в этом каталоге может находиться набор картинок. В нашем случае это будет восемь очень похожих, но все же разных изображений, о которых мы говорили в разделе «Планирование покадровой анимации». Затем мы создадим XML-файл, в котором определим список кадров (листинг 6.3). Этот XML-файл также должен будет находиться в подкаталоге /res/drawable.

Листинг 6.3. XML-файл, в котором определяется список кадров для анимации

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
      android:oneshot="false">
   <item android:drawable="@drawable/colored-ball1"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball2"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball3"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball4"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball5"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball6"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball7"
         android:duration="50" />
   <item android:drawable="@drawable/colored-ball8"
         android:duration="50" />
</animation-list>
```

Каждый кадр указывает на одно из изображений шарика, которые собраны вместе по их ID ресурса. Тег animation-list преобразуется в объект AnimationDrawable, представляющий собой коллекцию изображений. Затем потребуется задать этот Drawable как ресурс для фона ImageView, который использован в примере. Предположим, что XML-файл из нашего примера называется frame_animation.xml и находится в подкаталоге /res/drawable. Тогда следующий код позволит задать AnimationDrawable в качестве фона для ImageView:

```
view.setBackGroundResource(Resource.drawable.frame_animation);
```

При помощи этого кода Android узнает, что ID ресурса Resource.drawable.frame_animation соответствует XML-ресурсу. Исходя из этого, создает подходящий объект Java AnimationDrawable для данного ресурса и только потом задает такой ресурс в ка-

честве фона. После того как фон будет установлен, вы сможете получить доступ к объекту AnimationDrawable, применив get к объекту view таким образом:

```
Object backgroundObject = view.getBackground();
AnimationDrawable ad = (AnimationDrawable)backgroundObject;
```

Когда у вас будет AnimationDrawable, вы сможете использовать методы start() и stop() этого объекта, чтобы запускать и останавливать анимацию. Ниже показаны еще два важных метода, применяемых с этим объектом:

```
setOneShot();
addFrame(drawable, duration);
```

Метод setOneShot() один раз воспроизводит анимацию, а затем останавливает ее. Метод addFrame() добавляет новый кадр, используя объект Drawable, и устанавливает длительность ее воспроизведения. Функционально метод addFrame() напоминает XML-тег android:drawable.

Теперь соберем все это вместе, чтобы получить полный код для тестового приложения, в котором используется покадровая анимация (листинг 6.4).

Листинг 6.4. Полный код тестового приложения, в котором используется анимация

```
public class FrameAnimationActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.frame_animations_layout);
      this.setupButton();
   }

   private void setupButton()
   {
      Button b = (Button)this.findViewById(R.id.startFAButtonId);
      b.setOnClickListener(
         new Button.OnClickListener(){
            public void onClick(View v)
            {
               parentButtonClicked(v);
            }
         });
   }
   private void parentButtonClicked(View v)
   {
      animate();
   }
   private void animate()
   {
      ImageView imgView = (ImageView)findViewById(R.id.imageView);
      imgView.setVisibility(ImageView.VISIBLE);
      imgView.setBackgroundResource(R.drawable.frame_animation);

      AnimationDrawable frameAnimation =
```

```
            (AnimationDrawable) imgView.getBackground();
        if (frameAnimation.isRunning())
        {
            frameAnimation.stop();
        }
        else
        {
            frameAnimation.stop();
            frameAnimation.start();
        }
    }
} // eof-класс
```

Метод animate() располагает ImageView в актуальном явлении и устанавливает в качестве фона AnimationDrawable, идентифицируемый по ресурсу R.drawable.frame_animation. Затем код возвращает этот объект и запускает анимацию. Кнопка **Start/Stop** (Запустить/остановить) настроена так, что, если анимация воспроизводится, нажатие кнопки остановит процесс. А если анимация остановлена, то при нажатии кнопки начнется воспроизведение.

Обратите внимание: если установить для параметра OneShot в списке рисунков для анимации значение true, то анимация остановится после окончания первого воспроизведения. Но у нас в арсенале нет специального метода, который позволил бы определить, когда именно закончится воспроизведение. Хотя анимация и закончится после показа последней картинки, нам не хватает обратного вызова, который сообщил бы системе об этом. Следовательно, мы не можем непосредственно запустить следующее действие, которое было бы ответом на окончание воспроизведения анимации.

Если не учитывать этот недостаток, нарисовав серию изображений, вы можете создать впечатляющие визуальные эффекты. Но покадровая анимация — это все же очень незамысловатый прием.

Анимация шаблонов

Итак, покадровая анимация — это дешевый и сердитый способ добавления визуальных эффектов в программу Android. Анимация шаблонов лишь незначительно превосходит покадровую анимацию в сложности. Анимация шаблонов используется с ListView и GridView — двумя наиболее часто применяемыми в Android элементами управления. Если быть точными, при использовании анимации шаблонов визуальные эффекты связываются с самим методом, который применяется для отображения элементов в ListView или GridView. На самом деле анимация такого типа используется с любыми элементами управления, производными от ViewGroup.

Как было отмечено в начале этой главы, при анимации шаблонов к каждому из видов, входящему в состав анимируемого шаблона, применяется техника *построения промежуточных кадров* (tweening). В процессе построения промежуточных кадров ряд свойств вида изменяется с регулярными интервалами. Изменяя такую

матрицу несколькими способами, в виде можно реализовать эффекты масштабирования, вращения и перемещения (трансляции) элементов. Например, изменив прозрачность вида с 0 на 1, можно обеспечить так называемую ***альфа-анимацию.***

В этом разделе мы напишем простое приложение, при помощи которого сможем изучить и протестировать возможности анимации шаблонов, а также провести несколько экспериментов. Мы покажем, как ассоциировать `ListView` и анимацию с построением промежуточных кадров. Кроме того, коротко опишем интерполяторы: объясним, что это такое и какую роль они играют при анимации. В документации по SDK в разделе об интерполяторах есть некоторые пробелы, поэтому работа интерполяторов будет специально разъяснена на примерах с приведением соответствующего исходного кода. Мы также обсудим так называемый `LayoutAnimationController` — феномен, выступающий посредником между анимацией и `ViewGroup`.

Основные типы анимации с построением промежуточных кадров

Прежде чем перейти к написанию программы для тестирования различных типов анимации с построением промежуточных кадров, сделаем базовый обзор этих типов.

- *Масштабируемая анимация* — используется, когда вид нужно уменьшить или укрупнить по оси *X* или *Y*. Кроме того, можно указать осевую точку (pivot point), вокруг которой будет располагаться анимация.
- *Анимация вращения* — применяется для вращения вида вокруг осевой точки на определенное количество градусов.
- *Анимация трансляции* — используется для перемещения вида вдоль оси *X* или *Y*.
- *Альфа-анимация* — применяется для изменения степени прозрачности вида.

Все значения параметров, связанные с этими видами анимации, имеют разновидности `from` и `to`, так как вы должны указать начальные и конечные значения, соответствующие первой и последней стадиям показа анимации. Каждая анимация также может принимать в качестве аргументов такие параметры, как длительность (duration) и временной интерполятор (time interpolator). Мы рассмотрим интерполяторы в конце этой главы, когда будем говорить об анимации шаблонов. Пока остановимся на том, что интерполяторы определяют частоту изменения анимированного аргумента в процессе анимации.

Такие виды анимации задаются как XML-файлы, находящиеся в подкаталоге `/res/anim`. Этот вопрос будет подробно объяснен в тестовом приложении. Сейчас же обратите внимание на листинг 5.6 — этот небольшой пример поможет вам закрепить представление об описанных выше типах анимации.

Листинг 6.5. Масштабируемая анимация, заданная в XML-файле /res/anim/scale.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">
   <scale
         android:fromXScale="1"
```

```
        android:toXScale="1"
        android:fromYScale="0.1"
        android:toYScale="1.0"
        android:duration="500"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="100" />
</set>
```

Имея такой файл, вы сможете связать данную анимацию с шаблоном. Это означает, что анимация будет применяться к каждому из видов шаблона. В тестовом приложении, к которому мы вскоре обратимся, этот процесс рассмотрен более подробно.

ПРИМЕЧАНИЕ

Все перечисленные виды анимации представлены в виде классов Java и находятся в пакете android.view.animation. В документации Java, посвященной каждому из этих классов, описаны не только соответствующие методы Java, но и разрешенные XML-аргументы, которые соответствуют каждому типу анимации.

Итак, теперь вы достаточно подробно изучили базовые сведения о типах анимации и можете понять анимацию шаблонов. Перейдем к написанию тестового приложения, в котором используется данная анимация.

Подготовка тестовой программы для испытания анимации шаблона

Все концепции, связанные с анимацией шаблонов и рассмотренные выше, можно протестировать, реализовав в явлении простой набор `ListView`. К `ListView` можно прикрепить анимацию, которая будет применяться к каждому из видов шаблона.

Предположим, у нас есть масштабируемая анимация, в соответствии с которой вид растет от 0 до своего оригинального размера по оси *Y*. Такую анимацию можно связать с `ListView`. После этого `ListView` будет анимировать все элементы из списка, используя анимацию такого типа. Можно задать дополнительные параметры анимации, например анимирование списка по направлению сверху вниз или наоборот. Эти параметры указываются при помощи промежуточного класса, играющего роль посредника между отдельно взятой анимацией и списком.

Вы можете задать как отдельные анимированные элементы, так и необходимых посредников в XML-файлах, которые будут располагаться в подкаталоге `/res/anim`. Имея XML-файл с описанием посредников, можно использовать его информацию для ввода в `ListView` при определении его собственного XML-шаблона. Этот момент станет понятнее после изучения листингов с кодом, которые мы рассмотрим далее в данном разделе. Когда будет налажена работа по базовым параметрам, можно будет приступить к изменению отдельных элементов анимации, чтобы увидеть, как они влияют на отображение `ListView`.

В примерах будут рассмотрены масштабируемая анимация, анимация трансляции, анимация вращения и комбинированное использование анимации транс-

ляции и альфа-анимации. Если такая глобальная перспектива кажется вам немного туманной, просто внимательно следите за материалом: в конце этого раздела все встанет на свои места.

Прежде чем приступить к выполнению упражнения, посмотрим, как ListView будет выглядеть после завершения анимации (рис. 6.4).

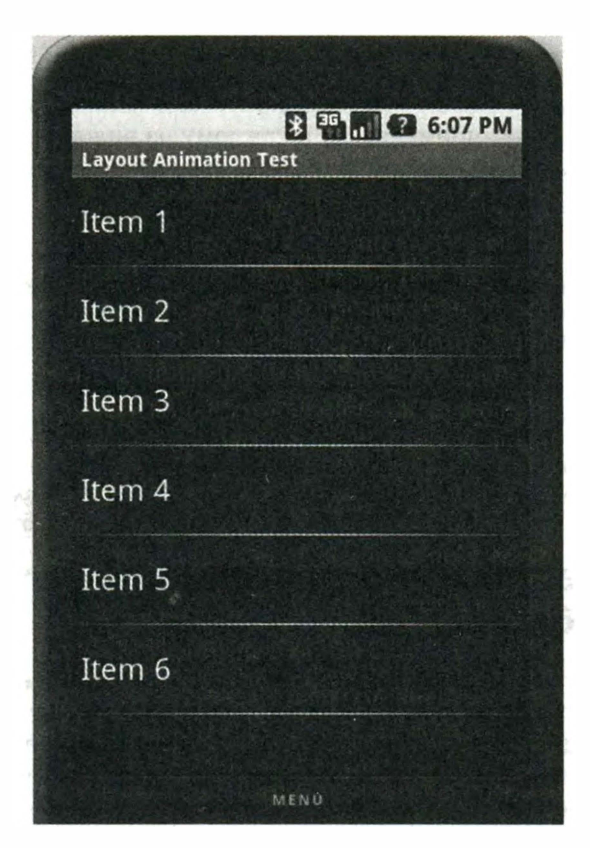


Рис. 6.4. ListView, анимацией которого мы будем заниматься

Создание явления и ListView

Начнем с создания XML-шаблона для ListView, показанного на рис. 6.4, так, чтобы можно было загрузить этот шаблон в базовое явление. В листинге 6.6 дан простой шаблон, в котором находится ListView. Этот файл нужно поместить в подкаталог /res/layout. Допустим, имя файла — list_layout.xml. В таком случае полный адрес файла будет иметь вид /res/layout/list_layout.xml.

Листинг 6.6. XML-файл шаблона, в котором определяется ListView

```
<?xml version="1.0" encoding="utf-8"?>
<!-- имя файла: /res/layout/list_layout.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

    <ListView
        android:id="@+id/list_view_id"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        />
</LinearLayout>
```

В листинге 6.6 показан простой LinearLayout, в котором находится только один ListView. Однако относительно определения ListView необходимо отметить одну деталь. Если вы будете работать с примерами для NotePad и других программ Android, то заметите, что ID для ListView обычно имеет вид @android:id/list. Мы уже упоминали в главе 3 о том, что ссылка @android:id/list указывает на ID, заранее заданный в пространстве имен Android. Вопрос заключается в следующем: когда нам использовать такой android:id, а когда ID, заданный нами, например @+id/list_view_id?

@android:id/list будет применяться только с явлением ListActivity. ListActivity предполагает, что ListView, обозначаемый таким заранее заданным ID, доступен для загрузки. В таком случае используется явление общего порядка, а не ListActivity. Это означает, что вам придется специально заполнить ListView. В результате мы можем свободно выбирать вид ID, при помощи которого будет представлен такой ListView. Правда, вы с полным правом можете воспользоваться и @android:id/list — ListActivity в данном контексте отсутствует и конфликтов не возникает.

Здесь необходимо сделать одно отступление, касающееся создания собственных ListView вне ListActivity. Обратите внимание: шаблон, который требуется для работы явления, у нас уже есть. Поэтому мы можем написать код для загрузки этого файла шаблона, чтобы в нем генерировался ваш пользовательский интерфейс (листинг 6.7).

Листинг 6.7. Код явления для анимирования шаблона

```
public class LayoutAnimationActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list_layout);
        setupListView();
    }
    private void setupListView()
    {
        String[] listItems = new String[] {
            "Item 1", "Item 2", "Item 3",
            "Item 4", "Item 5", "Item 6",
        };

        ArrayAdapter listItemAdapter =
```

```
        new ArrayAdapter(this
            ,android.R.layout.simple_list_item_1
            ,listItems);
    ListView lv = (ListView)this.findViewById(R.id.list_view_id);
    lv.setAdapter(listItemAdapter);
  }
}
```

Часть кода из листинга 6.7 является очевидной, а часть — нет. Первая часть кода просто загружает шаблон на основе сгенерированного ID R.layout.list_layout, относящегося к шаблону. Наша цель — взять ListView из этого шаблона и заполнить его шестью текстовыми элементами. Эти текстовые элементы загружаются в виде массива. Необходимо установить в ListView адаптер данных, чтобы ListView мог отобразить эти элементы.

Чтобы создать нужный адаптер, потребуется указать, как именно должен выглядеть каждый элемент, отображаемый в списке. Задайте один из готовых шаблонов, имеющихся в базе фреймворка Android. В нашем примере шаблон задается так:

```
android.R.layout.simple_list_item_1
```

Кроме того, такие элементы могут иметь следующие шаблоны видов:

```
simple_list_item_2
simple_list_item_checked
simple_list_item_multiple_choice
simple_list_item_single_choice
```

В документации по Android объяснено, как выглядит и действует каждый из этих шаблонов. Теперь вы можете активировать это явление из любого элемента меню вашего приложения, применив следующий код:

```
Intent intent = new Intent(inActivity,LayoutAnimationActivity.class);
inActivity.startActivity(intent);
```

Но, как и всегда при активации явления, вам потребуется зарегистрировать LayoutAnimationActivity в файле AndroidManifest.xml, чтобы система срабатывала на предыдущем этапе — при активации намерения. Код для этой цели выглядит так:

```
<activity android:name=". LayoutAnimationActivity"
 android:label="View Animation Test Activity"/>
```

Анимирование ListView

Вот мы и написали тестовое приложение (см. листинги 6.6 и 6.7). Теперь научимся применять к этому ListView конкретные варианты масштабируемой анимации. Рассмотрим, как применить такую анимацию к данному ListView в XML-файле (листинг 6.8).

Листинг 6.8. Задание масштабируемой анимации в XML-файле

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">
```

```
  <scale
      android:fromXScale="1"
      android:toXScale="1"
      android:fromYScale="0.1"
      android:toYScale="1.0"
      android:duration="500"
      android:pivotX="50%"
      android:pivotY="50%"
      android:startOffset="100" />
</set>
```

Такие файлы, в которых определяется анимация, находятся в подкаталоге /res/anim. Переведем эти атрибуты с XML на русский язык. Масштабы from и to указывают начальный и конечный коэффициенты увеличения (magnification factor). В данном случае увеличение по оси *X* имеет значение 1, которое впоследствии не изменяется. Но по оси *Y* мы имеем начальный коэффициент увеличения 0,1, который в итоге возрастает до 1,0. Иными словами, анимируемый объект при появлении имеет размер в 1/10 своего обычного размера, а затем растет, пока не увеличится в 10 раз. Для завершения операции масштабирования потребуется 500 миллисекунд. Эпицентром операции является 50%-ное увеличение по осям *X* и *Y*. Значение соответствует количеству миллисекунд, через которое должна начинаться анимация.

Родительский узел масштабируемой анимации указывает на набор элементов анимации (animation set), при помощи которого можно одновременно задействовать более одного анимационного эффекта. Мы также рассмотрим один из таких примеров. Но пока в наборе есть только один анимационный элемент.

Назовем этот файл scale.xml и расположим его в подкаталоге /res/anim. Пока вы не можете задать данный анимационный XML-файл в качестве аргумента ListView. Сначала для ListView потребуется другой XML-файл, который будет действовать как посредник между ListView и набором анимационных элементов. XML-файл, в котором дано такое опосредование, показан в листинге 6.9.

Листинг 6.9. Задание XML-файла для контроллера шаблона

```
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
  android:delay="30%"
  android:animationOrder="reverse"
  android:animation="@anim/scale" />
```

Этот XML-файл также потребуется сохранить в подкаталоге /res/anim. В нашем примере предположим, что файл называется list_layout_controller. Взглянув на такое определение, мы сразу видим, зачем нужен файл-посредник. В XML-файле указано, что анимация в списке должна осуществляться в обратном порядке и что анимирование каждого элемента будет начинаться с 30%-ной задержкой относительно срока воспроизведения анимации. Этот XML-файл также относится к конкретному файлу анимации — scale.xml. Кроме того, вместо имени файла в коде используется ссылка на ресурс @anim/scale.

Теперь у нас есть файлы, необходимые для ввода. Далее будет показано, как обновить XML-определение ListView, чтобы анимационный XML-файл мог действовать

в качестве аргумента. Сначала давайте посмотрим, какие XML-файлы у нас уже есть:

```
// конкретная масштабируемая анимация
/res/anim/scale.xml

// файл, опосредующий анимацию
/res/anim/list_layout_controller.xml

// файл шаблона вида явления
/res/layout/list_layout.xml
```

Имея эти файлы, мы должны изменить XML-файл шаблона list_layout.xml, чтобы ListView указывал на файл list_layout_controller.xml (листинг 6.10).

Листинг 6.10. Обновленный код для файла list_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <ListView
      android:id="@+id/list_view_id"
      android:persistentDrawingCache="animation|scrolling"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layoutAnimation="@anim/list_layout_controller" />
      />
</LinearLayout>
```

Измененные строки выделены полужирным шрифтом. android:layoutAnimation — это ключевой тег, указывающий на опосредующий XML-файл, в котором при помощи XML-тега layoutAnimation определяется контроллер шаблона (см. листинг 6.9). Тег layoutAnimation, в свою очередь, указывает на конкретный анимационный элемент — в данном случае это масштабируемая анимация, заданная в scale.xml. В Android также рекомендуется ставить тег persistentDrawingCache, предназначенный для оптимизации прокрутки и анимации. Более подробно этот тег описан в документации по Android SDK.

При дополнении файла list_layout.xml и приведении его в вид, показанный в листинге 6.10, плагин Eclipse ADT автоматически перекомпилирует пакет с учетом внесенных изменений. Если теперь запустить приложение, вы увидите, что к отдельным элементам, имеющимся на экране, будет применена масштабируемая анимация. Мы установили для длительности значение 500 миллисекунд — таким образом, вы будете четко видеть изменение масштаба при отрисовке каждого элемента.

Теперь мы готовы перейти к экспериментам с различными типами анимации. Начнем с альфа-анимации. Для этого создадим файл /res/anim/alpha.xml и вставим в него код из листинга 6.11.

Листинг 6.11. Файл alpha.xml для тестирования альфа-анимации

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="1000" />
```

Альфа-анимация управляет ослабеванием яркости цвета. В данном примере мы ставим для альфа-анимации задачу превратить элемент из невидимого в полноцветный за 1000 миллисекунд, или за 1 секунду. Длительность изменения должна составлять 1 секунду или более. В противном случае изменение цвета будет сложно различить.

В подобных случаях, когда изменяется анимация отдельно взятого элемента, потребуется изменить и опосредующий XML-файл (см. листинг 6.9), чтобы он указывал на новый анимационный файл. В следующем фрагменте показано, как изменить масштабируемую анимацию на альфа-анимацию:

```
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
   android:delay="30%"
   android:animationOrder="reverse"
   android:animation="@anim/alpha" />
```

Измененная строка XML-файла layoutAnimation выделена полужирным. Теперь рассмотрим анимацию, в которой одновременно изменяется и положение элемента, и цвет. В листинге 6.12 показан образец XML-файла для такой анимации.

Листинг 6.12. Совместное использование анимации трансляции и альфа-анимации с набором элементов

```
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:interpolator="@android:anim/accelerate_interpolator">
   <translate android:fromYDelta="-100%" android:toYDelta="0"
android:duration="500" />
   <alpha android:fromAlpha="0.0" android:toAlpha="1.0"
android:duration="500" />
</set>
```

Обратите внимание: в наборе элементов мы указали два типа анимации. Анимация трансляции перемещает текст по направлению сверху вниз в ту точку экрана, в которой текст находится на рисунке. Альфа-анимация изменяет градиент цвета с невидимого на видимый по мере того, как текст опускается на отведенное для него место на экране. Если установить для длительности анимации значение 500, пользователю будет удобно наблюдать за изменением цвета. Разумеется, нам вновь понадобится изменить опосредующий XML-файл layoutAnimation, поставив ссылку на новое имя файла. Предположим, файл с такой комбинированной анимацией будет называться /res/anim/translate-alpha.xml. В этом случае XML-файл с комбинированной анимацией приобретет следующий вид:

```
<layoutAnimation xmlns:android="http://schemas.android.com/apk/res/android"
   android:delay="30%"
   android:animationOrder="reverse"
   android:animation="@anim/translate-alpha" />
```

Теперь изучим на практике анимацию вращения (листинг 6.13).

Листинг 6.13. XML-файл анимации вращения

```
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromDegrees="0.0"
   android:toDegrees="360"
   android:pivotX="50%"
   android:pivotY="50%"
   android:duration="500" />
```

Код из листинга 6.13 позволяет вращать каждый текстовый элемент списка по полной окружности вокруг центральной точки текстового элемента. Длительность анимации 500 миллисекунд является достаточной, чтобы пользователь ясно увидел вращение. Как и в предыдущих случаях, для визуализации такого эффекта нужно изменить XML-файл контроллера шаблона и XML-файл шаблона `ListView`, а затем перезапустить приложение.

Итак, мы рассмотрели базовые концепции, связанные с анимацией шаблонов. В примерах мы создавали простой анимационный файл и связывали его с `ListView` при помощи файла-посредника `layoutAnimation` (XML). Это минимум, необходимый для создания эффектов анимации. Однако следует обсудить еще одну вещь, связанную с анимацией шаблонов, — интерполяторы.

Работа с интерполяторами

Интерполяторы сообщают анимации, как определенное свойство, например цвет, должно изменяться с течением времени. Будет ли это свойство изменяться линейно или экспоненциально? Будет ли изменение начинаться быстро, но замедляться по мере приближения к концу? Рассмотрим альфа-анимацию, которая была описана в листинге 6.11:

```
<alpha xmlns:android="http://schemas.android.com/apk/res/android"
   android:interpolator="@android:anim/accelerate_interpolator"
   android:fromAlpha="0.0" android:toAlpha="1.0" android:duration="1000" />
```

Данная анимация идентифицирует интерполятор, который должен использоваться с нею, в данном случае — `accelerate_interpolator`. Это соответствующий объект Java, в котором дается определение интерполятора. Отметим также, что мы обозначили этот интерполятор при помощи ссылки на ресурс. Это означает, что у нас должен быть файл, соответствующий `anim/accelerate_interpolator` и описывающий, какой вид имеет этот объект Java и какие дополнительные параметры он может принимать (в частности, значение регистра). Рассмотрим определение XML-файла для `@android:anim/accelerate_interpolator`:

```
<accelerateInterpolator
   xmlns:android="http://schemas.android.com/apk/res/android"
   factor="1" />
```

Этот XML-файл будет находиться в следующем подкаталоге в пакете Android:

```
/res/anim/accelerate_interpolator.xml
```

XML-тег accelerateInterpolator соответствует объекту Java со следующим именем:

```
android.view.animation.AccelerateInterpolator
```

В разделе документации Java, касающемся этого класса, вы можете просмотреть доступные XML-теги. Задача интерполятора — дать коэффициент умножения с учетом временного интервала на основе гиперболической кривой. Это видно из исходного кода интерполятора:

```
public float getInterpolation(float input)
{
   if (mFactor == 1.0f)
   {
      return (float)(input * input);
   }
   else
   {
      return (float)Math.pow(input, 2 * mFactor);
   }
}
```

Каждый интерполятор использует метод getInterpolation по-своему. В данном случае, если интерполятор имеет коэффициент умножения 1.0, будет возвращен квадрат коэффициента. В ином случае будет возвращена степень ввода, которая в дальнейшем будет умножена на коэффициент. То есть, если коэффициент равен 1.5, вместо квадратичной функции будет использоваться кубическая.

В системе поддерживаются следующие интерполяторы:

```
AccelerateDecelerateInterpolator
AccelerateInterpolator
CycleInterpolator
DecelerateInterpolator
LinearInterpolator
AnticipateInterpolator
AnticipateOvershootInterpolator
BounceInterpolator
OvershootInterpolator
```

Чтобы оценить, насколько гибкими могут быть эти интерполяторы, рассмотрим BounceInterpolator, который раскачивает (bounce) объект, то есть перемещает его вперед-назад до окончания следующей анимации:

```
public class BounceInterpolator implements Interpolator {
   private static float bounce(float t) {
      return t * t * 8.0f;
   }

   public float getInterpolation(float t) {
      t *= 1.1226f;
      if (t < 0.3535f) return bounce(t);
      else if (t < 0.7408f) return bounce(t - 0.54719f) + 0.7f;
```

```
        else if (t < 0.9644f) return bounce(t - 0.8526f) + 0.9f;
        else return bounce(t - 1.0435f) + 0.95f;
    }
}
```

Работа этих интерполяторов описана по следующей ссылке: http://developer.android.com/reference/android/view/animation/package-summary.html.

В документации Java по каждому из этих классов также указываются XML-теги, применяемые для управления классом. Но из документации сложно понять, какие именно функции выполняет каждый интерполятор. Лучше всего попробовать интерполятор на практике и посмотреть, что получится. Для поиска исходного кода можно воспользоваться следующей ссылкой: http://android.git.kernel.org/?p=platform%2Fframeworks%2Fbase.git&a=search&h=HEAD&st=grep&s=BounceInterpolator.

На этом мы завершаем обсуждение анимации шаблонов и переходим к третьему разделу, где будет рассказано, как анимировать виды средствами программирования.

Анимация видов

Вы уже знакомы с покадровой анимацией и анимацией шаблонов и можете вплотную заняться анимацией видов — наиболее сложным типом анимации. Анимация видов применяется к любому виду путем управления матрицей преобразований (transformation matrix), используемой для отображения вида.

Начнем этот раздел с краткого введения в анимацию видов. Затем покажем код тестового приложения (в этом приложении можно будет поэкспериментировать с анимацией видов) и приведем несколько примеров анимации видов. Потом объясним, как использовать с анимацией видов объект Camera (этот объект не имеет ничего общего с обычной камерой, установленной в устройстве, — феномен является полностью графическим). Наконец, мы подробно изучим, как работать с матрицами преобразований.

Общие сведения об анимации видов

Когда вид отображается на поверхности представления (presentation surface) в Android, он пропускается через матрицу преобразований (transformation matrix). В графических приложениях матрицы преобразований используются для изменения вида тем или иным способом. В ходе процесса берется совокупность входных значений (input set), координат пикселов и цветовых комбинаций, которая преобразуется в новую совокупность таких значений и комбинаций. По завершении преобразования вы увидите изображение, в котором изменены размер, положение, ориентация или цвет.

Все эти преобразования производятся математически — совокупность входных координат умножается тем или иным способом при помощи матрицы преобразований для получения нового набора координат. Изменяя матрицу преобразований, вы воздействуете на внешний вид элемента на экране. Матрица, которая не изме-

няет элемент при умножении, называется *матрицей тождественного преобразования* (identity matrix). Обычно работа начинается с матрицы тождественного преобразования, после чего производятся изменения размера, положения и ориентации элемента. Затем берется матрица, полученная в итоге, и используется для рисования вида.

В Android при применении матрицы преобразований к виду вы получаете возможность зарегистрировать вместе с видом анимационный объект. Анимационный объект будет снабжен обратным вызовом, который позволяет получить от вида актуальную матрицу и определенным образом изменить ее, чтобы получить новый вид. Ниже мы детально проработаем этот процесс.

Сначала спланируем пример, в котором будет анимироваться вид. Начнем с явления, в котором будет размещен `ListView` с несколькими элементами — приблизительно так же, как и в примере с анимацией шаблонов. Затем создадим кнопку в верхней части экрана, при нажатии которой будет начинаться анимация `ListView` (рис. 6.5). На экране появятся и кнопка, и `ListView`, но анимация пока происходить не будет. Нажатие кнопки будет запускать анимацию.

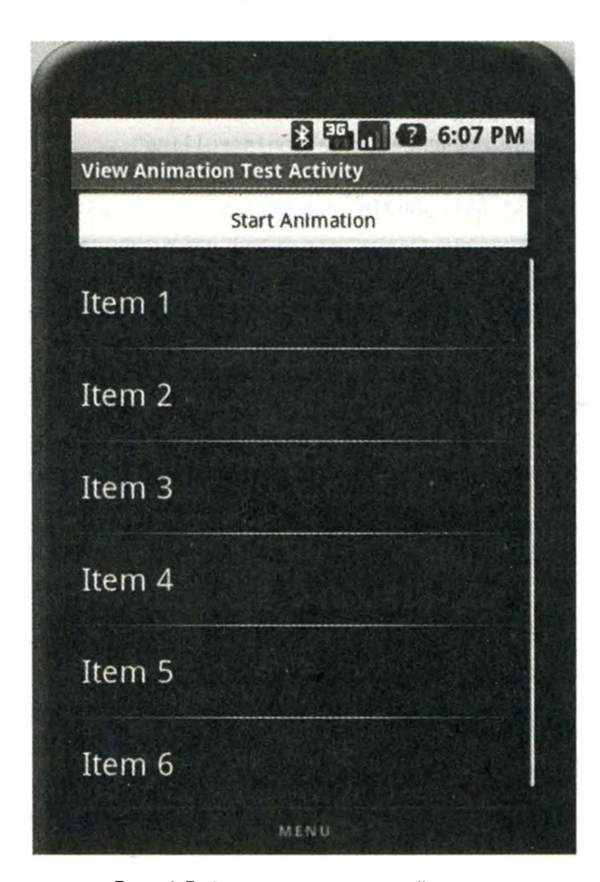


Рис. 6.5. Явление с анимацией видов

В этом примере при нажатии кнопки **Start Animation** (Запустить анимацию) уменьшенный вид должен появиться в тексте экрана, а потом увеличиваться, пока не займет все отведенное для него место. Мы покажем, как написать код для этой операции. В листинге 6.14 приведен XML-файл шаблона, который можно использовать с этим явлением.

Листинг 6.14. XML-файл шаблона для явления с анимацией вида

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Этот файл находится в at /res/layout/list_layout.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button
   android:id="@+id/btn_animate"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Start Animation"
/>
<ListView
   android:id="@+id/list_view_id"
   android:persistentDrawingCache="animation|scrolling"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
/>
</LinearLayout>
```

Обратите внимание, что название и расположение файла указаны для справки в верхней части XML-файла. Данный шаблон состоит из двух элементов: кнопки, которая называется `btn_animate` и используется для анимирования вида, и шаблона `ListView`, называющегося `list_view_id`.

Теперь у нас есть шаблон явления и можно создать само явление, в котором будет показан вид и вставлена кнопка **Start Animation** (Запустить анимацию) (листинг 6.15).

Листинг 6.15. Код для явления с анимацией вида до начала анимации

```
public class ViewAnimationActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.list_layout);
      setupListView();
      this.setupButton();
   }
   private void setupListView()
   {
      String[] listItems = new String[] {
```

```
        "Item 1", "Item 2", "Item 3",
        "Item 4", "Item 5", "Item 6",
   };

      ArrayAdapter listItemAdapter =
         new ArrayAdapter(this
               ,android.R.layout.simple_list_item_1
               ,listItems);
      ListView lv = (ListView)this.findViewById(R.id.list_view_id);
      lv.setAdapter(listItemAdapter);
   }
   private void setupButton()
   {
      Button b = (Button)this.findViewById(R.id.btn_animate);
      b.setOnClickListener(
         new Button.OnClickListener(){
            public void onClick(View v)
            {
               // animateListView();
            }
         });
   }
}
```

Код явления с анимацией вида (листинг 6.15) очень напоминает код явления с анимацией шаблона (см. листинг 6.7). Мы также загрузили вид и установили ListView, который будет содержать шесть текстовых элементов. Кнопка создается таким образом, чтобы при нажатии она вызывала animateListView(). Но на данном этапе деактивируйте эту часть (то есть закомментируйте ее) до тех пор, пока не приведете в рабочее состояние базовый пример.

Для активации этого явления его сначала нужно зарегистрировать его в файле AndroidManifest.xml:

```
<activity android:name=".ViewAnimationActivity"
   android:label="View Animation Test Activity">
```

Когда регистрация будет закончена, данное анимационное явление можно будет активировать из любого элемента меню вашего приложения, выполнив следующий код:

```
Intent intent = new Intent(this, ViewAnimationActivity.class);
startActivity(intent);
```

При запуске программы пользовательский интерфейс будет иметь вид как на рис. 6.5.

Добавление анимации

В данном случае наша задача — расположить анимацию так, чтобы ListView выглядел как на рис. 6.5. Для этого требуется класс, производный от android.view.animation.Animation. При наличии такого класса нужно переопределить метод

applyTransformation, изменив таким образом матрицу преобразований. Производный класс назовем ViewAnimation. Сделав подобный класс, мы сможем произвести над классом ListView операцию следующего рода:

```
ListView lv = (ListView)this.findViewById(R.id.list_view_id);
lv.startAnimation(new ViewAnimation());
```

Теперь двинемся дальше. Рассмотрим исходный код ViewAnimation и обсудим, какую именно анимацию мы хотим получить (листинг 6.16).

Листинг 6.16. Код класса ViewAnimation

```
public class ViewAnimation extends Animation
{
    public ViewAnimation2(){}

    @Override
    public void initialize(int width, int height, int parentWidth,
                                        int parentHeight)
  {
      super.initialize(width, height, parentWidth, parentHeight);
      setDuration(2500);
      setFillAfter(true);
      setInterpolator(new LinearInterpolator());
  }
  @Override
  protected void applyTransformation
          (float interpolatedTime, Transformation t)
  {
    final Matrix matrix = t.getMatrix();
    matrix.setScale(interpolatedTime, interpolatedTime);
  }
}
```

Initialize — это метод обратного вызова, сообщающий параметры вида. Здесь же инициализируются любые возможные параметры анимации. В данном примере мы задали длительность анимации, равную 2500 миллисекундам (2,5 секунды). Мы также указали, что анимационный эффект должен сохраняться и после завершения анимации — для этого параметр FillAfter принимает значение true. К тому же, как было указано выше, мы установили линейный интерполятор. Это означает, что анимация изменяется от начала к концу постепенно. Все данные свойства входят в состав базового класса android.view.animation. Animation

Основная часть анимации осуществляется в рамках метода applyTransformation. Фреймворк Android для обеспечения анимационного эффекта вызывает этот метод снова и снова. Каждый раз метод interpolatedTime имеет иное значение. Значения находятся в диапазоне от 0 до 1, в зависимости от того, в какой момент 2,5-секундной анимации происходит инициализация. Если interpolatedTime имеет значение 1, анимация завершена.

Наша цель — изменить матрицу преобразований, которая доступна через объект преобразования t, относящийся к методу applyTransformation. Сначала берем матрицу и определенным образом изменяем ее. Когда вид будет отрисован, задействуется новая матрица. Виды методов, применимых к объекту Matrix, перечис-

лены в документации по API для android.graphics.Matrix: **http://developer.android.com/reference/android/graphics/Matrix.html**.

В листинге 6.16 дан код для изменения матрицы:

```
matrix.setScale(interpolatedTime, interpolatedTime);
```

Метод setScale принимает два параметра: коэффициент масштабирования (scaling factor) по оси *X* и аналогичный коэффициент по направлению *Y*. Поскольку диапазон значений interpolatedTime расположен в пределах от 0 до 1, это значение можно непосредственно использовать как коэффициент масштабирования. В начале анимации коэффициент масштабирования равен 0 по осям *X* и *Y*. По завершении анимации этот коэффициент будет равен 1 — следовательно, изображение примет свой истинный размер. В результате анимации ListView вырастает от почти невидимого до полного размера.

В листинге 6.17 показан весь исходный код явления ViewAnimationActivity, в состав которого входит и анимация.

Листинг 6.17. Код для явления с анимацией вида

```
public class ViewAnimationActivity extends Activity {

   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.list_layout);
      setupListView();
      this.setupButton();
   }
   private void setupListView()
   {
      String[] listItems = new String[] {
         "Item 1", "Item 2", "Item 3",
         "Item 4", "Item 5", "Item 6",
      };

      ArrayAdapter listItemAdapter =
         new ArrayAdapter(this
           ,android.R.layout.simple_list_item_1
           ,listItems);
      ListView lv = (ListView)this.findViewById(R.id.list_view_id);
      lv.setAdapter(listItemAdapter);
   }
   private void setupButton()
   {
      Button b = (Button)this.findViewById(R.id.btn_animate);
      b.setOnClickListener(
         new Button.OnClickListener(){
         public void onClick(View v)
         {
            animateListView();
         }
```

```
        });
    }
    private void animateListView()
    {
        ListView lv = (ListView)this.findViewById(R.id.list_view_id);
        lv.startAnimation(new ViewAnimation());
    }
}
```

После запуска кода из листинга 6.17 вы заметите нечто странное. ListView начинает вырастать не из середины экрана, а из верхнего левого угла. Причина этого заключается в том, что работа матрицы начинается именно с верхнего левого угла. Для получения желаемого эффекта необходимо сначала переместить весь вид, чтобы центр вида совпал с центральной точкой анимации (верхним левым углом). Затем нужно применить матрицу и вернуть вид обратно в центр.

Это делается при помощи следующего кода:

```
final Matrix matrix = t.getMatrix();
matrix.setScale(interpolatedTime, interpolatedTime);
matrix.preTranslate(-centerX, -centerY);
matrix.postTranslate(centerX, centerY);
```

Методы preTranslate и postTranslate настраивают матрицу до начала операции масштабирования и после ее окончания. Это равнозначно трем парным матричным преобразованиям. Код:

```
matrix.setScale(interpolatedTime, interpolatedTime);
matrix.preTranslate(-centerX, -centerY);
matrix.postTranslate(centerX, centerY);
```

означает

```
перейди к другому центру
масштабируй
вернись на прежний центр
```

Вот код метода преобразований, необходимого для нужного нам эффекта:

```
protected void applyTransformation(float interpolatedTime, Transformation t)
{
    final Matrix matrix = t.getMatrix();
    matrix.setScale(interpolatedTime, interpolatedTime);
    matrix.preTranslate(-centerX, -centerY);
    matrix.postTranslate(centerX, centerY);
}
```

pre и post будут применяться по такому принципу снова и снова. Аналогичный результат достигается и при помощи других методов класса Matrix, но описанная выше техника наиболее распространена, не говоря уже о том, как краток сам код.

Важнее отметить, что класс Matrix позволяет не только масштабировать вид, но и поворачивать его по окружности при помощи методов translate, а также изменять его ориентацию посредством методов rotate. Поэкспериментируйте с тремя этими методами и посмотрите, как в итоге будет выглядеть анимация.

Все анимационные примеры, приведенные в предыдущем разделе «Анимация шаблонов», реализуются внутри системы при помощи методов из класса Matrix.

Использование класса Camera для создания эффекта глубины изображения в 2D

В графическом пакете Android есть еще один класс, относящийся к анимации, вернее, к преобразованиям. Он называется Camera. Этот класс используется для создания эффекта глубины путем проецирования двухмерного изображения, движущегося в трехмерном пространстве, на двухмерную плоскость. Например, мы можем взять ListView и переместить его «за экран» на 10 пикселов по оси *Z* и повернуть его же по оси *Y* на 30°. Ниже приведен пример использования матрицы с классом Camera:

```
Camera camera = new Camera();
..
protected void applyTransformation(float interpolatedTime, Transformation t)
{
   final Matrix matrix = t.getMatrix();
   camera.save();
   camera.translate(0.0f, 0.0f, (1300 - 1300.0f * interpolatedTime));
   camera.rotateY(360 * interpolatedTime);
   camera.getMatrix(matrix);

   matrix.preTranslate(-centerX, -centerY);
   matrix.postTranslate(centerX, centerY);
   camera.restore();
}
```

Этот код анимирует шаблон ListView. Сначала вид отодвигается на 1300 пикселов назад от плоскости экрана по оси *Z*, а затем возвращается обратно на экран, где значение по оси *Z* равно 0. При этом код также вызывает вращение вида с 0 до 360° по оси *Y*. Рассмотрим код, относящийся к такому способу действия, и изучим следующий метод:

```
camera.translate(0.0f, 0.0f, (1300 - 1300.0f * interpolatedTime));
```

Данный метод приказывает объекту camera транслировать вид так, чтобы при значении interpolatedTime, равном 0 (в начале анимации), значение *z* было равно 1300. По мере воспроизведения анимации, значение *z* будет уменьшаться и уменьшаться, пока значение interpolatedTime не станет равно 1, а значение *z* — 0.

Метод camera.rotateY(360 * interpolatedTime) использует функцию трехмерного вращения по оси, предоставляемую в методе Camera. В момент начала воспроизведения анимации это значение будет равно 0. В конце воспроизведения анимации оно будет равно 360.

Метод camera.getMatrix(matrix) принимает операции, которые к данному моменту были выполнены методом Camera, и применяет эти операции к полученной матрице. Когда код осуществит эти операции, у матрицы будут в наличии все трансляции, необходимые для завершения эффекта Camera. Теперь Camera находится за пределами картинки, так как все необходимые операции внедрены в саму матрицу.

К матрице можно применять pre и post, чтобы создавалась иллюзия движения ее центра вперед и назад. Наконец Camera возвращается в исходное состояние, сохраненное нами ранее.

Вставив этот код в наш пример, вы увидите, что ListView движется, вращаясь вокруг своей оси, к экрану — как и планировалось при проектировании анимации.

Рассказывая об анимации видов, мы показали, как можно анимировать любой вид путем дополнения класса Animation и применения внесенных изменений к виду. Класс Animation позволяет не только работать с матрицами (как непосредственно, так и через класс Camera), но и идентифицировать различные этапы анимации. Эти аспекты будут рассмотрены ниже.

Изучение класса AnimationListener

В Android используется интерфейс прослушивания AnimationListener, предназначенный для отслеживания событий, связанных с анимацией (листинг 6.18). Для получения данных о таких событиях интерфейс AnimationListener внедряется в систему и настраивается на прием информации от экземпляра класса Animation.

Листинг 6.18. Реализация интерфейса AnimationListener

```
public class ViewAnimationListener
implements Animation.AnimationListener {

   private ViewAnimationListener(){}

   public void onAnimationStart(Animation animation)
   {
      Log.d("Animation Example", "onAnimationStart");
   }
   public void onAnimationEnd(Animation animation)
   {
      Log.d("Animation Example", "onAnimationEnd");
   }
   public void onAnimationRepeat(Animation animation)
   {
      Log.d("Animation Example", "onAnimationRepeat");
   }
}
```

Класс ViewAnimationListener просто регистрирует сообщения. Можно обновить метод animateListView в примере с анимацией видов (см. листинг 6.17), чтобы при работе учитывался слушатель анимации:

```
private void animateListView()
{
   ListView lv = (ListView)this.findViewById(R.id.list_view_id);
   ViewAnimation animation = new ViewAnimation();
   animation.setAnimationListener(new ViewAnimationListener());
 lv.startAnimation(animation);
}
```

Несколько замечаний о матрицах преобразований

В этой главе было указано, что при преобразовании видов и анимации матрицы имеют ключевое значение. Ниже мы кратко рассмотрим некоторые важнейшие методы класса Matrix. Далее перечислены основные операции, производимые с матрицами:

```
matrix.reset();
matrix.setScale();
matrix.setTranslate()
matrix.setRotate();
matrix.setSkew();
```

Первая операция сбрасывает значение матрицы до матрицы идентичности, которая не вызывает никаких изменений вида. SetScale отвечает за изменение размера, SetTranslate — за изменение положения элемента и имитацию движения, SetRotate — за изменение ориентации, а SetSkew — за деформацию вида.

Можно связывать матрицы или перемножать их, чтобы достигать кумулятивного эффекта одиночных преобразований. Рассмотрим следующий пример, где m1, m2 и m3 — матрицы идентичности:

```
m1.setScale();
m2.setTranlate()
m3.concat(m1,m2)
```

Преобразование вида матрицей m1 и последующее преобразование полученного вида при помощи m2 эквивалентно преобразованию того же вида матрицей m3. Обратите внимание — методы set заменяют предыдущие преобразования, а m3.concat(m1,m2) не равно m3.concat(m2,m1).

Мы уже изучали принцип, используемый в рамках методов preTranslate и postTranslate для воздействия на преобразование матрицы. На самом деле методы pre и post используются не только с translate — отдельные версии pre и post применяются с каждым из методов преобразований из группы set. В итоге preTranslate, например m1.preTranslate(m2), эквивалентен m1.concat(m2,m1).

Аналогично метод m1.postTranslate(m2) эквивалентен m1.concat(m1,m2).

И, само собой разумеется, что код

```
matrix.setScale(interpolatedTime, interpolatedTime);
matrix.preTranslate(-centerX, -centerY);
matrix.postTranslate(centerX, centerY);
```

эквивалентен

```
Matrix matrixPreTranslate = new Matrix();
matrixPreTranslate.setTranslate(-centerX, -centerY);

Matrix matrixPostTranslate = new Matrix();
matrixPostTranslate.setTranslate(cetnerX, centerY);

matrix.concat(matrixPreTranslate,matrix);
matrix.postTranslate(matrix,matrixpostTranslate);
```

Резюме

В этой главе был показан интересный способ усовершенствования пользовательских интерфейсов — путем улучшения их анимацией. Мы рассмотрели все основные типы анимации, поддерживаемые в Android, в том числе покадровую анимацию, анимацию шаблонов и анимацию видов. Мы также изучили вспомогательные сущности, связанные с анимацией, — интерполяторы и матрицы преобразований.

Теперь, имея такие базовые знания, вы можете самостоятельно рассмотреть API, имеющиеся в Android SDK, и опробовать образцы определений XML с различными видами анимации. Мы ненадолго вернемся к вопросам, связанным с анимацией, в главе 10, где покажем, как рисовать и анимировать изображения при помощи OpenGL.

А теперь вам предстоит узнать много нового о службах, используемых в Android. В главе 7 мы поговорим о службах, основанных на местоположении, и о службах безопасности, а в главе 8 — о службах, которые относятся к работе с протоколом HTTP.

7 Изучение вопросов безопасности и служб, основанных на местоположении

В этой главе мы поговорим о применяемой в Android модели обеспечения безопасности приложений, а также о службах, работа которых зависит от местоположения (location-based services). Хотя две эти темы совсем и не связаны друг с другом, необходимо понять принципы обеспечения безопасности, прежде чем перейти к работе со службами, зависящими от местоположения.

В начале главы изучается безопасность — фундаментальная составляющая платформы Android. В Android безопасность обеспечивается на всех стадиях жизненного цикла приложения — от соблюдения политик на этапе разработки до проверки граничных условий во время исполнения программы. Мы изучим применяемую в Android архитектуру системы безопасности и научимся создавать надежно защищенные приложения.

Следующая часть главы посвящена службам, работающим на основе местоположения. В состав таких служб входит один или несколько очень интересных элементов инструментария Android SDK. Данная часть SDK предоставляет разработчикам API, при помощи которых можно отображать карты и управлять ими, в реальном времени получать данные о местоположении устройства, пользоваться преимуществами других интереснейших функций. Прочитав эту часть книги, вы окончательно убедитесь, что Android — по-настоящему интересная штука.

Итак, начнем с изучения модели обеспечения безопасности в Android.

Модель обеспечения безопасности в Android

Безопасность в Android обеспечивается в ходе развертывания приложений (application deployment) и их выполнения. Что касается развертывания, приложения Android должны быть отмечены цифровыми сертификатами (иметь цифровые подписи), чтобы их можно было установить в устройство. Относительно выполнения — Android запускает любое приложение в рамках отдельного процесса, каждый из которых имеет уникальный и постоянный пользовательский ID (присваиваемый в ходе установки). Таким образом, вокруг процесса создается граница

и каждое из приложений не может напрямую получить доступ к данным другого приложения. Более того, в Android применяется модель выделения декларативных полномочий (declarative permission model), защищающая конфиденциальные компоненты системы (например, список контактов).

В нескольких следующих разделах мы обсудим эти темы. Но прежде, чем начать, сделаем обзор некоторых концепций, связанных с безопасностью.

Обзор концепций, связанных с безопасностью

Выше мы сказали о том, что каждое приложение в Android должно иметь цифровой сертификат. Одно из достоинств, связанных с этим требованием, заключается в том, что приложение нельзя обновить до неофициальной версии, выпущенной кем-либо, кроме разработчика программы. Например, если мы выпускаем программу, а вы сами пишете для нее обновление, то вы не сможете обновить написанное нами приложение до вашей версии — разумеется, если вы не заполучите каким-либо образом наш сертификат и связанный с ним пароль. Итак, что же такое цифровая подпись приложения и как происходит цифровая сертификация приложения?

Цифровой сертификат — это компонент программы, в котором содержатся данные о разработчике — название вашей компании, адрес и т. д. К важнейшим атрибутам цифрового сертификата относятся его подпись и открытый/закрытый ключ. Открытый и закрытый ключи вместе также называются парой ключей. Обратите внимание — в данном случае цифровые сертификаты используются для создания цифровых подписей файлов APK, но их можно применять и в других целях (например, для зашифрованной связи). Цифровой сертификат можно получить в доверенном центре сертификации (trusted certificate authority, CA) или сгенерировать самостоятельно при помощи специальных инструментов (например, `keytool`), о которых мы вскоре поговорим. Цифровые сертификаты хранятся в *хранилищах ключей* (keystore). Они содержат списки цифровых сертификатов, каждый из которых имеет псевдоним (alias). Эти псевдонимы используются для ссылки на ключи, находящиеся в хранилище.

Чтобы у приложения Android появилась цифровая подпись, нужны три вещи: цифровой сертификат, файл APK и утилита, которой известно, как применить подпись цифрового сертификата к APK-файлу. Мы пользуемся бесплатной утилитой, входящей в состав дистрибутива инструментария для разработки на Java (JDK), которая называется jarsigner. Это инструмент для работы с командной строкой, умеющий создавать цифровые сертификаты для файлов JAR.

Теперь рассмотрим, как подписать файл APK и оснастить его цифровым сертификатом.

Подписывание приложений для развертывания

Чтобы установить приложение Android в устройство, сначала нужно оснастить пакет Android (файл APK) цифровой подписью сертификата. Сертификат может быть самозаверяющим (self-signed), тогда вам не придется приобретать сертификат в центре сертификации, таком как VeriSign.

Чтобы дать приложению цифровую подпись, необходимую для развертывания, выполните три шага. Сначала сгенерируйте сертификат при помощи keytool (или подобного инструмента). Затем примените инструмент jarsigner (или его аналог), чтобы снабдить файл APK подписью сгенерированного сертификата. На третьем этапе выровняйте компоненты вашего приложения в имеющихся границах памяти (memory boundaries) для более эффективного использования ресурсов при эксплуатации программы в устройстве. Обратите внимание: в процессе разработки подписыванием вашего файла APK и выравниванием в памяти (memory alignment) занимается ADT-плагин для Eclipse, а затем программа развертывается в эмуляторе. Более того, сертификат, по умолчанию используемый для выставления цифровой подписи при разработке, невозможно применять для развертывания продукта в реальном устройстве.

Генерирование самозаверяющегося сертификата при помощи keytool

Утилита keytool управляет базой данных, в которой содержатся закрытые ключи и соответствующие им сертификаты стандарта X.509. Эта утилита входит в состав инструментария JDK и располагается в каталоге JDK bin. Если в главе 2 вы во всех случаях правильно изменяли PATH, то каталог JDK bin уже должен быть в PATH.

В этом подразделе будет рассказано, как создать хранилище ключей с одной записью, которая позже понадобится нам для подписи файла APK Android. Чтобы создать запись в хранилище ключей, выполните следующее.

1. Создайте каталог для хранилища ключей, например c:\android\release\.
2. Откройте окно инструментов (о том, что такое окно инструментов, подробно рассказано в главе 2) и выполните служебную программу keytool с параметрами, приведенными в листинге 7.1.

Листинг 7.1. Генерация записи для хранилища ключей при помощи keytool

```
keytool -genkey -v -keystore "FULL PATH OF release.keystore FILE FROM STEP 1"
-alias androidbook -storepass paxxword -keypass paxxword -keyalg RSA
-validity 14000
```

Аргументы, которые могут быть переданы keytool, приведены в табл. 7.1.

Таблица 7.1. Аргументы, которые могут быть переданы keytool

Аргумент	Описание
genkey	Приказывает keytool сгенерировать пару из открытого и закрытого ключа
v	Приказывает keytool не выводить подробный отчет (verbose output) в ходе генерирования ключа
keystore	Путь к базе данных хранилища ключей (в данном случае — к файлу)
alias	Уникальное имя записи, содержащейся в хранилище ключей. Псевдоним (alias) в дальнейшем будет использоваться для ссылки на эту запись
storepass	Пароль к хранилищу ключей
keyalg	Алгоритм
validity	Срок действия

keytool подскажет о том, что нужно указать пароли, перечисленные в табл. 7.1, если вы не введете их в командную строку. Если не вы один пользуетесь данным компьютером, то безопаснее будет не указывать -storepass и -keypass в командной строке, а вводить их в поля подсказок, предлагаемые keytool. Команда из листинга 7.1 создает файл базы данных хранилища ключей в каталоге, предназначенном для таких хранилищ. Базой данных будет файл release.keystore. Параметр validity записи составит 14 000 дней (примерно 38 лет, период довольно долгий). Сейчас мы объясним, почему так делается. В документации по Android рекомендуется указывать достаточно долгий срок действия, чтобы гарантированно превысить срок жизненного цикла приложения. Рекомендуется устанавливать срок действия не менее 25 лет. Более того, если вы собираетесь опубликовать приложение в Android Market (http://www.android.com/market/), сертификат должен быть годен как минимум до 22 октября 2033 года. Android Market проверяет каждое загружаемое приложение, чтобы гарантировать, что оно будет действовать как минимум до этого срока.

Возвращаясь к keytool, отметим, что аргумент alias — это уникальное имя, присваиваемое записи, которая находится в базе данных хранилища ключей. Это имя можно использовать для ссылки на запись. При запуске команды из листинга 7.1, keytool задаст вам некоторые вопросы (рис. 7.1), после чего создаст базу данных хранилища ключей и запись.

```
C:\WINDOWS\system32\cmd.exe

C:\Program Files\Java\jre1.6.0_07\bin>keytool -genkey -v -keystore c:\android\re
lease\release.keystore -alias androidbook -storepass paxxword -keypass paxxword
-keyalg RSA -validity 14000
What is your first and last name?
  [Unknown]:  sayed
What is the name of your organizational unit?
  [Unknown]:  IT
What is the name of your organization?
  [Unknown]:  sayedhashimi
What is the name of your City or Locality?
  [Unknown]:  Jacksonville
What is the name of your State or Province?
  [Unknown]:  FL
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=sayed, OU=IT, O=sayedhashimi, L=Jacksonville, ST=FL, C=US correct?
  [no]:  yes

Generating 1,024 bit RSA key pair and self-signed certificate (SHA1withRSA) with
 a validity of 14,000 days
        for: CN=sayed, OU=IT, O=sayedhashimi, L=Jacksonville, ST=FL, C=US
[Storing c:\android\release\release.keystore]

C:\Program Files\Java\jre1.6.0_07\bin>_
```

Рис. 7.1. Дополнительные вопросы, задаваемые keytool

Вот и готов цифровой сертификат, при помощи которого можно подписать файл APK. Чтобы это сделать, воспользуйтесь инструментом jarsigner. Ниже описано, как это выполнить.

Использование инструмента jarsigner при подписывании файла APK

В предыдущем подразделе keytool создавал цифровой сертификат, являющийся для инструмента jarsigner одним из параметров. Вторым параметром jarsigner является

тот пакет, который необходимо подписать. Чтобы сгенерировать пакет Android, воспользуйтесь утилитой **Export Unsigned Application Package**, входящей в состав плагина ADT программы Eclipse. Чтобы перейти к этой утилите, откройте в Eclipse проект Android, щелкнув на нем правой кнопкой мыши, выберите **Android Tools** (Инструменты Android) и здесь выберите **Export Unsigned Application Package**. При запуске **Export Unsigned Application Package** будет сгенерирован файл APK, который не будет подписан отладочным сертификатом (debug certificate). Чтобы посмотреть, как это работает, запустите утилиту **Export Unsigned Application Package** с одним из ваших проектов Android и сохраните где-нибудь сгенерированный файл APK. В данном примере мы будем использовать каталог хранилища ключей, созданный нами ранее, и сгенерируем файл APK, называемый `c:\android\release\myapp.apk`.

Когда будут готовы файл APK и запись в хранилище ключей, запустите инструмент `jarsigner`, чтобы подписать файл APK (листинг 7.2). В случае необходимости при таком запуске используйте полные имена путей к файлам хранилища ключей и APK.

Листинг 7.2. Использование jarsigner для подписывания файла APK

```
jarsigner -keystore "PATH TO YOUR release.keystore FILE" -storepass paxxword
-keypass paxxword "PATH TO YOUR APK FILE" androidbook
```

Чтобы подписать файл APK, вы передаете расположение хранилища ключей, пароль к этому хранилищу, пароль к закрытому ключу, путь к файлу APK и псевдоним записи хранилища ключей. Чтобы запустить инструмент `jarsigner`, нужно открыть либо окно инструментов (об этом рассказано в главе 2), либо командное окно (окно терминала). В нем вам потребуется или перейти в каталог `JDK bin`, или убедиться, что ваш каталог `JDK bin` указан в полном абсолютном пути к системному каталогу (system path).

Выше мы говорили о том, что приложение в Android должно быть подписано цифровым сертификатом, чтобы злоумышленник не смог обновить ваше приложение своей версией. Чтобы этот механизм работал, обновления любой программы Android должны иметь такую же цифровую подпись, как и оригинал. Если дать программе другую подпись, Android будет считать две версии двумя различными программами.

Выравнивание приложения в памяти при помощи zipalign

Разумеется, вы хотите, чтобы ваша программа при работе в устройстве использовала ресурсы памяти максимально эффективно. Если во время исполнения в программе есть незаархивированные данные (например, определенные типы изображений или файлы с данными), Android может поместить эти данные прямо в память, воспользовавшись вызовом `mmap()`. Однако чтобы этот механизм работал, данные должны быть выровнены по 4-байтной границе памяти. Процессоры на устройствах Android являются 32-битными, а 32 бит равно 4 байт. Вызов `mmap()` делает информацию вашего файла APK неотличимой на вид от памяти, но, если данные не выровнены по 4-битной границе, вызов `mmap()` не может сработать таким образом и во время выполнения программы необходимо осуществить дополнительное копирование данных. Инструмент `zipalign`, который находится в каталоге инструментов

в Android SDK, просматривает ваше приложение и постепенно выравнивает по границе 4 байт все незаархивированные данные. От этого размер приложения может увеличиться, но ненамного. Чтобы выровнять файл APK, введите в окно инструментов следующую команду (рис. 7.2):

```
zipalign -v 4 infile.apk outfile.apk
```

```
C:\WINDOWS\system32\cmd.exe
C:\android\release>zipalign -v 4 weightgravity.apk weightgravityaligned.apk
Verifying alignment of weightgravityaligned.apk (4)...
      50 META-INF/MANIFEST.MF (OK - compressed)
     400 META-INF/ANDROIDB.SF (OK - compressed)
     818 META-INF/ANDROIDB.RSA (OK - compressed)
    1536 res/drawable/icon.png (OK)
    4951 res/layout/main.xml (OK - compressed)
    5399 AndroidManifest.xml (OK - compressed)
    6008 resources.arsc (OK)
    7093 classes.dex (OK - compressed)
Verification succesful

C:\android\release>_
```

Рис. 7.2. Работа с zipalign

Обратите внимание: `zipalign` проверяет выравнивание при создании выровненного файла. Если нужно перезаписать имеющийся файл `outfile.apk`, можно использовать параметр `-f`. Итак, чтобы гарантировать, что имеющийся файл правильно выровнен, используйте `zipalign` следующим образом:

```
zipalign -c -v 4 filename.apk
```

Очень важно: выравнивание делается *после* подписывания, иначе из-за подписывания выравнивание снова может сбиться. Это еще не означает, что программа откажет, но она может начать расходовать больше памяти, чем это необходимо.

После того как файл APK будет подписан и выровнен в памяти, его можно будет вручную установить в эмулятор при помощи инструмента `adb`. Итак, запустим эмулятор. Мы пока не испробовали еще один способ запуска — перейдите в меню **Window** (Окно) программы Eclipse, выберите в нем **Android SDK** и диспетчер виртуальных устройств Android (**AVD Manager**). Отобразится окно, где будут показаны доступные виртуальные устройства Android. Выберите устройство, с которым собираетесь работать в эмуляторе, и нажмите кнопку **Start** (Пуск). Эмулятор запустится, не копируя при этом каких-либо ваших проектов, находящихся в разработке в Eclipse. Теперь откройте окно инструментов и запустите инструмент `adb` установочной командой:

```
adb install "ЭТО ПУТЬ К ФАЙЛУ APK"
```

По некоторым причинам попытка может не удаться. Правда, если неудача и произойдет, то, скорее всего, из-за того, что отладочная версия вашего приложения уже установлена в эмуляторе, и поэтому вы получите ошибку «Такая программа уже установлена». В таком случае отладочная версия программы деинсталлируется следующей командой:

```
adb uninstall imjapaketa
```

Обратите внимание, что при деинсталляции приложения аргументом является имя пакета приложения, а не имя файла APK. Имя пакета определяется в файле AndroidManifest.xml установленного приложения. Во втором случае вы можете использовать приведенную ниже команду, в которой -r означает деинсталляцию приложения, но с сохранением ее данных на устройстве (или эмуляторе):

```
adb install -r " ЭТО ПУТЬ К ФАЙЛУ APK "
```

Перейдем к тому, как подписывание действует на процесс обновления приложения.

Установка обновлений программы и цифровое подписывание

Ранее мы упоминали о том, что сертификат имеет дату истечения срока и что Google рекомендует указывать в таком качестве даты из далекого будущего в расчете на то, что программе предстоит многократно обновиться. Что же происходит, когда истекает срок действия цифрового сертификата? Не перестанет ли программа Android после этого запускаться? К счастью, нет. Android проверяет дату окончания срока действия сертификата только в процессе установки программы. Установленное приложение будет работать и после того, как срок действия сертификата закончится.

А как быть с обновлениями? К сожалению, после истечения срока действия сертификата обновить программу будет невозможно. Иными словами, Google подсказывает нам, что сертификат должен иметь срок действия, который гарантированно превысит срок жизненного цикла приложения. После истечения срока действия сертификата Android уже не установит никаких его обновлений. Единственный выход — создать новую программу (с новым названием пакета) и подписать ее новым сертификатом. Как видите, очень важно правильно задать срок действия сертификата уже при его создании.

Итак, мы изучили проблемы безопасности на этапах развертывания и установки программы. Теперь давайте рассмотрим, как обеспечивается безопасность программ Android во время их исполнения.

Проверка безопасности системы во время исполнения

В Android безопасность системы во время исполнения обеспечивается на уровне процессов и на операционном уровне. На уровне процессов Android исключает возможность непосредственного доступа одного приложения к данным другого приложения. Для этого каждая программа выполняется как отдельный процесс, под уникальным и постоянным пользовательским ID. На операционном уровне Android определяет список защищенных функций и ресурсов. Чтобы ваша программа могла получить доступ к этой информации, нужно добавить в файл AndroidManifest.xml один или более запросов на разрешение (permission request). В рамках приложения вы также можете настраивать специальные права доступа.

В следующих подразделах мы поговорим об обеспечении безопасности на границе процессов, а также о том, как объявлять и использовать заданные права доступа. Кроме того, мы обсудим вопросы, связанные с определением специальных прав доступа и оптимизации их в конкретном приложении. Начнем с изучения того, как в Android обеспечивается безопасность на границе процессов.

Безопасность на границе процессов

В отличие от рабочей среды локального компьютера, где большинство программ выполняются под одинаковым пользовательским ID, каждое приложение Android обычно работает под собственным ID. Благодаря этому Android создает изоляционную границу (isolation boundary) вокруг каждого процесса. Таким образом, одно приложение не может получить непосредственный доступ к данным другого.

Хотя вокруг каждого процесса и есть граница, несколько приложений, разумеется, могут совместно использовать одни и те же данные. Но совместная работа с данными должна быть явной (explicit). Иными словами, путь к данным другого приложения лежит через его компоненты. Например, можно сделать запрос к поставщику содержимого другого приложения, активировать явление в другом приложении или установить контакт со службой другого приложения (об этом мы поговорим в главе 8). Все эти функции позволяют использовать методы, предназначенные для совместного использования данных между несколькими приложениями, но это делается явно, так как вы не обращаетесь к основной базе данных, файлам и т. д.

Итак, на уровне процессов система безопасности в Android построена просто и ясно. Но дело становится гораздо интереснее, когда разговор заходит о защите ресурсов (например, контактных данных) и собственных компонентов. Для обеспечения такой защиты в Android задается схема прав доступа (permission scheme). Изучим ее подробнее.

Определение и использование прав доступа

В Android определяется схема прав доступа, предназначенная для защиты ресурсов и функций устройства. Например, по умолчанию программа не может получить доступ к списку контактов, делать телефонные звонки и т. д. Чтобы защитить пользователя от вредоносных программ, приложения Android должны запрашивать право доступа, если им требуется воспользоваться защищенной функцией или защищенным ресурсом. Как вы вскоре увидите, запросы прав доступа направляются к файлу описания. В процессе установки APK-инсталлятор либо выдает такие права, либо отказывает в них, в зависимости от цифровой подписи файла APK и/или реакции пользователя. Если право доступа не получено, любая попытка выполнить соответствующую функцию или получить к ней доступ будет пресечена.

В табл. 7.2 перечислены часто используемые функции и права доступа, которые нужны для пользования ими. Обратите внимание: мы еще не знакомились с некоторыми из перечисленных функций, их предстоит изучить ниже (в этой главе или в следующих).

Таблица 7.2. Функции, ресурсы и права доступа, требуемые для работы с ними

Функция/ресурс	Необходимое право доступа	Описание
Камера	android.permission.CAMERA	Открывает доступ к камере, установленной на устройстве
Интернет	android.permission.INTERNET	Позволяет устанавливать сетевые соединения
Контактные данные пользователя	android.permission.READ_CONTACTS android.permission.WRITE_CONTACTS	Позволяет читать контактные данные пользователя или изменять их
Календарные данные пользователя	android.permission.READ_CALENDAR android.permission.WRITE_CALENDAR	Позволяет читать данные из календаря пользователя или изменять их
Запись аудио	android.permission.RECORD_AUDIO	Позволяет записывать аудио
GPS-данные о местоположении	android.permission.ACCESS_FINE_LOCATION	Открывает доступ к мелкоструктурной информации о местоположении, в том числе к данным GPS
Данные Wi-Fi о местоположении	android.permission.ACCESS_COARSE_LOCATION	Открывает доступ к крупноструктурной информации о местоположении, в том числе к данным Wi-Fi
Информация о батарее питания	android.permission.BATTERY_STATS	Открывает доступ к информации о состоянии элемента питания
Bluetooth	android.permission.BLUETOOTH	Открывает доступ к сопряженным устройствам, оснащенным Bluetooth

Полный список прав доступа приведен по следующей ссылке: http://developer.android.com/reference/android/Manifest.permission.html.

Разработчики программ могут запрашивать права доступа, добавляя записи в файл AndroidManifest.xml. В листинге 7.3 показано, как получить доступ к камере устройства, а также получить право чтения списка контактов и календаря.

Листинг 7.3. Права доступа в файле AndroidManifest.xml

```
<manifest ... >
   <application>
      ...
   </application>
   <uses-permission android:name="android.permission.CAMERA" />
   <uses-permission android:name="android.permission.READ_CONTACTS"/>
   <uses-permission android:name="android.permission.READ_CALENDAR" />
</manifest>
```

Обратите внимание, что права доступа можно либо внести в файл AndroidManifest.xml вручную, либо воспользоваться специальным редактором (manifest editor). Такой редактор описаний всегда запускается, когда вы открываете (двойным щелчком) файл описания. В редакторе описаний есть раскрывающийся список, в котором содержится полный набор прав доступа. Этот список избавляет вас от ошибок. На рис. 7.3 показано, как попасть в список прав доступа, выбрав в редакторе вкладку Permissions (Права доступа).

Теперь вы знаете, что в Android определяется набор прав доступа, предназначенных для защиты набора функций и ресурсов. Подобным образом можно задавать и оптимизировать специальные права доступа, которые будут действовать именно в вашей программе. Рассмотрим, как это работает на практике.

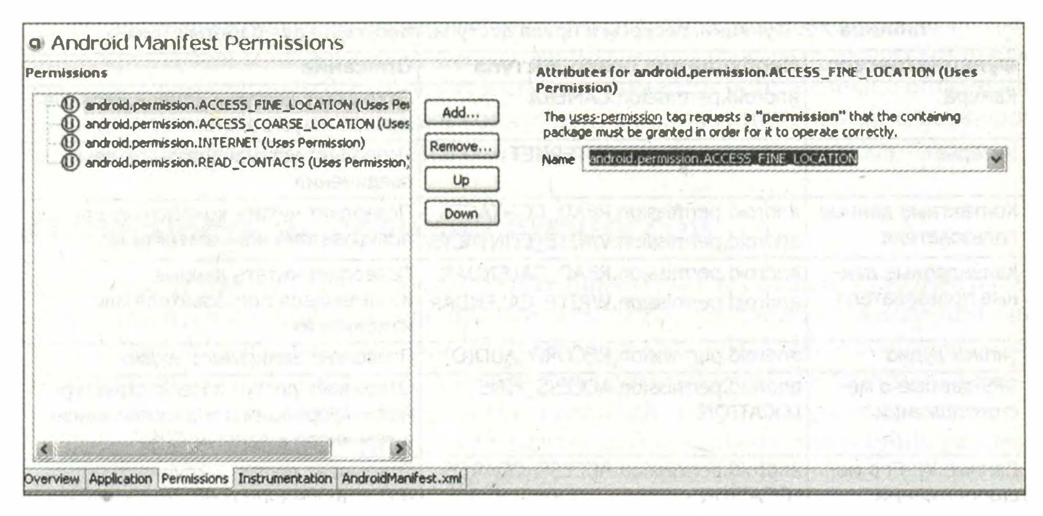


Рис. 7.3. Инструмент для редактирования файлов описаний Android в программе Eclipse

Специальные права доступа

В каждом конкретном приложении Android можно задавать специальные права доступа (custom permissions). Например, если вы хотите запретить определенным пользователям запускать некоторые явления вашей программы, это можно сделать при помощи специального дополнительного права доступа. Чтобы пользоваться специальными правами доступа, нужно сначала объявить их в файле описания `AndroidManifest.xml`. После того как право доступа будет задано, к нему можно обращаться в процессе определения компонентов (component definition). Рассмотрим, как это работает.

Создадим программу, содержащую такое явление, которое не могут запускать некоторые пользователи. Чтобы запустить это явление, пользователь должен иметь специальное право доступа. Когда в программе появляется подобное привилегированное явление, вы можете написать клиент, знающий, как вызвать такое явление.

Начнем с создания проекта, в котором будет специальное право доступа и привилегированное явление. Откройте Eclipse IDE и выберите **New ▸ New Project ▸ Android Project** (Создать ▸ Новый проект ▸ Проект Android). Появится диалоговое окно **New Android Project** (Новый проект Android). Назовите этот проект `CustomPermission`, установите переключатель в положение **Create new project in workspace** (Создать новый проект в рабочей области) и установите флажок **Use default location** (Использовать расположение, заданное по умолчанию). В качестве названия приложения укажите `Custom Permission`, имени пакета — `com.cust.perm`, имени явления — `CustPermMainActivity` и выберите параметр **Build Target** (Построить цель). Для завершения создания проекта нажмите кнопку **Finish** (Готово). В созданном проекте будет только что заданное вами явление. Оно будет использоваться как стандартное (основное) явление. Создадим также так называемое *привилегированное явление*, для работы с которым требуется специальное право доступа. В Eclipse IDE перейдите к пакету `com.cust.perm`, создайте класс `PrivActivity`, суперклассом которого будет `android.app.Activity`, и скопируйте код из листинга 7.4.

Листинг 7.4. Класс PrivActivity

```
package com.cust.perm;

import android.app.Activity;
import android.os.Bundle;
import android.view.ViewGroup.LayoutParams;
import android.widget.LinearLayout;
import android.widget.TextView;

public class PrivActivity extends Activity
{

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      LinearLayout view = new LinearLayout(this);

      view.setLayoutParams(new LayoutParams(
            LayoutParams.FILL_PARENT, LayoutParams.WRAP_CONTENT));
      view.setOrientation(LinearLayout..HORIZONTAL);

      TextView nameLbl = new TextView(this);

      nameLbl.setText("Hello from PrivActivity");
      view.addView(nameLbl);

      setContentView(view);

   }
}
```

Как видите, в PrivActivity нет ничего сверхъестественного. Мы просто хотели показать, как защитить это явление, используя специальное право доступа, а затем вызвать явление с помощью клиента. Если клиент сработает правильно, на экране отобразится текст Hello from PrivActivity. Теперь, имея явление, которое нужно защитить, можно приступить к созданию прав доступа к нему.

Чтобы создать специальное право доступа, его нужно задать в файле AndroidManifest.xml. Это лучше всего сделать в редакторе описаний. Дважды щелкните на файле AndroidManifest.xml, а затем выберите вкладку **Permissions** (Права доступа). Нажмите кнопку **Add** (Добавить), выберите **Permission** (Право доступа), а затем щелкните на **OK**. Редактор создаст новое пустое право доступа. Заполните это право доступа, установив для него атрибуты, как это показано на рис. 7.4. Заполните поля справа, и если заголовок слева так и не изменится (останется **Permissions** (Права доступа)), щелкните на нем правой кнопкой мыши — название должно замениться на одно из имен, указанных справа.

Как видно из рис. 7.4, право доступа имеет название (**Name**), обозначение (**Label**), пиктограмму (**Icon**), относится к определенной группе (**Permission group**). У него также есть описание (**Description**) и уровень защиты (**Protection level**). Эти свойства описаны в табл. 7.3.

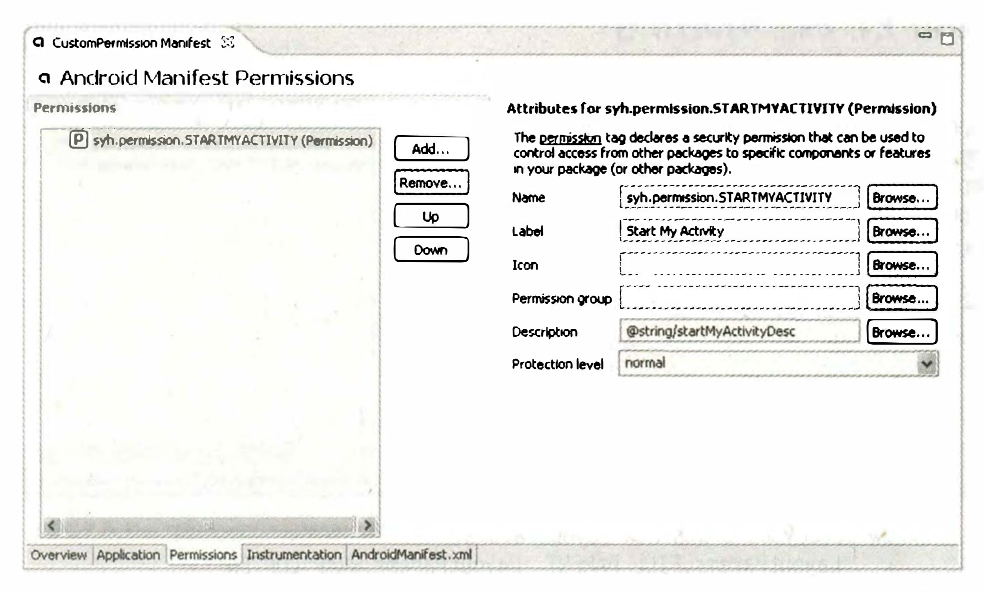


Рис. 7.4. Объявление специального права доступа при помощи редактора описаний

Таблица 7.3. Атрибуты права доступа

Атрибут	Необходимость	Описание
android:name	Да	Название права доступа. Обычно следует придерживаться принятой в Android схемы наименований (*.permission.*)
android:protectionLevel	Да	Определяет «степень риска», связанную с правом доступа. Может иметь одно из следующих значений: normal; dangerous; signature; signatureOrSystem. В зависимости от применяемого уровня защиты, система может предпринимать различные действия, определяя, следует давать право доступа или нет. Normal означает, что риск при выдаче права доступа невелик, и от этого не будет вреда системе, пользователю или другим программам. Dangerous говорит о том, что давать такое право доступа очень рискованно — перед предоставлением такого права система, скорее всего, потребует от пользователя ввода определенной информации. Signature сообщает Android, что право доступа может предоставляться только программам, имеющим такую же цифровую подпись, как и программа, которая объявила право доступа. SignatureOnSystem говорит Android, что право доступа может предоставляться только программам с одинаковой цифровой подписью или классам пакетов Android. Этот уровень защиты предусмотрен для особых случаев, при которых в работу вовлечено несколько производителей, обязанных совместно использовать функции с применением образа системы

Атрибут	Необходимость	Описание
android: permissionGroup	Нет	Можно объединить права доступа в группу, но именно со специальными правами доступа этого лучше избегать. Если вы все же хотите воспользоваться этим свойством, примените android.permission-group.SYSTEM_TOOLS
android:label	Нет	Хотя это свойство и не является обязательным, используйте его для краткого описания права доступа
android:description	Нет	Хотя это свойство и не является обязательным, применяйте его для более подробного описания принципа действия права доступа и того, какой именно аспект системы им защищается
android:icon	Нет	Права доступа могут быть связаны с определенными пиктограммами, которые не относятся к числу ресурсов (например, @drawable/myicon)

Итак, у нас есть специальное право доступа. Далее мы сообщаем системе, что явление PrivActivity может запускаться только такими приложениями, которые имеют право доступа syh.permission.STARTMYACTIVITY. Можно установить для явления требуемое право доступа, добавив при определении явления атрибут android:permission в файле AndroidManifest.xml. Чтобы вы могли запускать явление, к нему также нужен соответствующий фильтр намерений. Обновите файл AndroidManifest.xml, вставив в него код из листинга 7.5.

Листинг 7.5. Файл AndroidManifest.xml для проекта со специальными правами доступа

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.cust.perm"
      android:versionCode="1"
      android:versionName="1.0.0">
   <application android:icon="@drawable/icon"
               android:label="@string/app_name">
         <activity android:name=".CustPermMainActivity"
                  android:label="@string/app_name">
            <intent-filter>
               <action android:name="android.intent.action.MAIN" />
               <category android:name=
                    „android.intent.category.LAUNCHER" />
            </intent-filter>
         </activity>
   <activity android:name="PrivActivity"
android:permission="syh.permission.STARTMYACTIVITY">
      <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
   </activity>
</application>

<permission
```

```
android:protectionLevel="normal"
android:label="Start My Activity"
android:description="@string/startMyActivityDesc"
android:name="syh.permission.STARTMYACTIVITY"></permission>

   <uses-sdk android:minSdkVersion="2" />
</manifest>
```

В листинге 7.5 требуется добавить к числу строковых ресурсов строковую константу с именем startMyActivityDesc. Чтобы гарантировать успешную компиляцию листинга 7.5, добавьте в файл res/values/strings.xml следующий строковый ресурс:

```
<string name="startMyActivityDesc">Allows starting my activity</string>
```

Теперь запустите проект в эмуляторе. Хотя основное явление и не выполняет никаких задач, нужно сначала опробовать приложение на эмуляторе, а потом написать клиент для привилегированного явления.

Теперь напишем клиент для явления. В Eclipse IDE выполните команду **New ▸ New Project ▸ Android Project** (Создать ▸ Новый проект ▸ Проект Android). В качестве имени проекта введите ClientOfCustomPermission, установите переключатель в положение **Create new project in workspace** (Создать новый проект в рабочей области) и установите флажок **Use default location** (Использовать расположение, заданное по умолчанию). Назовите приложение Client Of Custom Permission, имя пакета будет com.client.cust.perm, а имя явления — ClientCustPermMainActivity. После этого нажмите **Build Target** (Построить цель). Для завершения создания проекта щелкните на кнопке **Finish** (Готово).

Затем мы напишем явление, в котором будет отображаться кнопка. При ее нажатии мы будем вызывать привилегированное явление. Скопируйте шаблон, приведенный в листинге 7.6, в файл main.xml только что созданного проекта.

Листинг 7.6. Файл Main.xml для проекта клиента

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <Button android:id="@+id/btn"
   android:text="Launch PrivActivity"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content" />
</LinearLayout>
```

Как видите, в XML-файле шаблона задана одна кнопка, на которой написано **Launch PrivActivity** (Запустить PrivActivity). Теперь напишем явление, которое будет обрабатывать событие нажатия кнопки и запускать привилегированное явление. Скопируйте код из листинга 7.7 в класс ClientCustPermMainActivity.

Листинг 7.7. Измененное явление ClientCustPermMainActivity

```
package com.client.cust.perm;
// это файл ClientCustPermMainActivity.java

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class ClientCustPermMainActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);

  Button btn = (Button)findViewById(R.id.btn);
  btn.setOnClickListener(new OnClickListener(){

    @Override
    public void onClick(View arg0) {

      Intent intent = new Intent();

      intent.setClassName("com.cust.perm",
                  "com.cust.perm.PrivActivity");
      startActivity(intent);
    }});
  }
}
```

В соответствии с листингом 7.7 вы получаете ссылку на кнопку, определенную в файле `main.xml`, а затем приводите в готовность слушатель событий щелчков (on-click). Когда кнопка будет активирована, мы создадим новое намерение, а затем установим имя класса явления, которое хотим запустить. В данном случае мы собираемся запустить `com.cust.perm.PrivActivity` из пакета `com.cust.perm`.

Все, чего нам еще не хватает, — это запись `uses-permission`, добавляемая в файл описания и сообщающая среде исполнения Android, что нужно запустить `syh.permission.STARTMYACTIVITY`. Замените в проекте клиента имеющийся файл описания на код из листинга 7.8.

Листинг 7.8. Файл описания клиента

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.client.cust.perm"
    android:versionCode="1"
    android:versionName="1.0.0">
  <application android:icon="@drawable/icon"
```

```
            android:label="@string/app_name">
        <activity android:name=".ClientCustPermMainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

    <uses-permission android:name="syh.permission.STARTMYACTIVITY">
    </uses-permission>
    <uses-sdk android:minSdkVersion="2" />
</manifest>
```

Из листинга 7.8 видно, что мы добавили запись uses-permission для запроса специального права доступа, который необходим для запуска явления PrivActivity, реализованного нами в этом проекте.

Теперь мы сможем развернуть клиентский проект в эмуляторе, а затем выбрать кнопку **Launch PrivActivity** (Запустить Priv Activity). После нажатия этой кнопки вы должны увидеть текст "Hello from PrivActivity".

После успешного вызова привилегированного явления выполните удаление записи uses-permission из файла описания проекта клиента и повторно разверните проект в эмуляторе. После этого убедитесь, что при попытке нажать кнопку, активирующую привилегированное явление, система выдает ошибку. Обратите внимание — LogCat отобразит исключение отказа в праве выполнения действия (permission-denial exception).

Итак, мы изучили, как в Android работают специальные права доступа. Очевидно, что они могут использоваться не только с явлениями. Права доступа, как содержащиеся в системе, так и заданные вами, можно применять и к любым другим типам компонентов в Android. Важный случай — права доступа к URI (уникальным идентификаторам ресурсов) будет рассмотрен в следующем подразделе.

Права доступа к URI и работа с ними

Часто поставщики содержимого (мы изучили их в главе 3) должны контролировать доступ на более сложном уровне, чем просто «все или ничего». К счастью, в Android предусмотрен соответствующий механизм. Возьмем, например, приложения к электронным письмам. Возможно, такое приложение должно быть считано другим явлением, чтобы можно было отобразить информацию, содержащуюся в приложении. Но другое явление не должно получать доступ ко всем данным электронной почты — более того, явлению не стоит предоставлять доступ даже ко всем приложениям. Вот здесь нам и пригодятся права доступа к URI.

При активации другого явления и передаче URI приложение может указать, что предоставляет права доступа только пересылаемому URI. Эту задачу выполняет метод grantUriPermission(), передающий в качестве аргумента Intent.FLAG_GRANT_READ_URI_PERMISSION или Intent.FLAG_GRANT_WRITE_URI_PERMISSION.

Работа со службами, основанными на местоположении

Службы Android, работа которых связана с местоположением устройства, функционируют на базе двух основных интерфейсов: картографического API и API местоположения. Оба этих интерфейса изолированы друг от друга, то есть относятся к собственным пакетам. Например, картографический пакет называется `com.google.android.maps`, а пакет местоположения — `android.location`. Картографический интерфейс в Android обслуживает функции отображения карты и работы с ней. Например, в его состав входят функции увеличения и панорамирования, с его помощью можно изменять режим работы карты (допустим, со спутникового изображения на просмотр изображений улиц), добавлять на карту собственные данные и т. д. Другими важнейшими элементами являются данные глобальной системы определения местоположения (GPS) и данные о местоположении в реальном времени. Данные обоих видов обрабатываются средствами пакета местоположения (location package).

Эти API получают доступ к данным, расположенным в любой точке Интернета, чтобы активировать службы, которые размещены на серверах Google. Следовательно, для их работы требуется возможность соединения по Интернету. Кроме того, Google определяет условия использования, которые вы принимаете, прежде чем сможете начать разработку приложений, использующих такие картографические службы Android. Внимательно читайте условия; Google накладывает определенные ограничения на использование служебных данных. Например, информация о местоположении может использоваться в интересах отдельно взятого абонента, но применение ее в некоторых коммерческих целях ограничено. Например, они не могут использоваться в программах для автоматического управления транспортными средствами. Условия будут представлены вам, когда вы запросите ключ к картографическому API (`map-api key`).

В этом разделе мы подробно изучим все упомянутые здесь пакеты. Начнем с картографического API. Здесь мы покажем, как вы можете использовать карты в своих программах. Как вы вскоре убедитесь, работа с картами в Android сводится к применению элемента управления `MapView`, входящего в состав пользовательского интерфейса, и класса `MapActivity` в дополнение к картографическим API, интегрированным в службу Google Maps. Мы также покажем, как добавлять на отображаемые карты вашу собственную информацию. Поговорив о картах, мы углубимся в работу со службами, работающими на базе местоположения устройства. Такие службы дополняют концепции, связанные с картографированием. Мы покажем, как использовать класс Android `Geocoder` и службу `LocationManager`. Кроме того, мы коснемся проблем, связанных с потоками. Они важны при работе с перечисленными API.

Пакет Mapping

Как было сказано выше, картографические API являются одним из компонентов служб Android, работающих на основе местоположения устройства. В пакете `Mapping`

содержится весь код, необходимый для отображения карты на экране, управления работой пользователя с картой (например, увеличением и уменьшением масштаба), отображения поверх карты специально добавляемых данных и т. д. Работа с этим пакетом начинается с отображения карты. Для этого используется класс `MapView`. Однако прежде, чем использовать этот класс, нужно выполнить некоторую подготовительную работу. В частности, перед применением `MapView` вы должны получить в Google ключ `map-api key`. Этот *ключ к картографическому API* обеспечивает взаимодействие между Android и службами Google Maps — в ходе такого взаимодействия вы получаете картографические данные. Ниже описано, как получить ключ `map api-key`.

Получение ключа к картографическому API в Google

При работе с ключом `map-api key` прежде всего необходимо усвоить, что вам нужно два ключа: один — для разработки программы в эмуляторе, другой — для внедрения программы в устройство. Причина в том, что сертификаты, используемые при получении картографических ключей для двух экземпляров программы — в эмуляторе и в устройстве, — будут различаться (мы говорили об этой проблеме в начале данной главы).

Допустим, в ходе разработки плагин ADT генерирует файл APK и развертывает его в эмуляторе. Поскольку файл APK должен быть подписан сертификатом, плагин ADT использует в ходе разработки отладочный сертификат. При окончательном внедрении программы для подписывания файла APK лучше применять самозаверяющий сертификат. Есть и хорошие новости: можно получить два ключа `map-api` — для разработки и для окончательного внедрения — и поменять ключи перед экспортированием окончательного релиза.

Для получения ключа `map-api` нужен сертификат, который будет использоваться для подписывания вашего приложения (напоминаем, что на этапе разработки плагин ADT использует для подписывания приложения отладочный сертификат и выполняет эту задачу за вас до самого развертывания программы в эмуляторе). Итак, вы получите «отпечаток» MD5 вашего сертификата, а затем введите его на сайте Google, чтобы система сгенерировала для вас соответствующий ключ для `map-api`.

Сначала найдите отладочный сертификат, который генерируется и поддерживается Eclipse. Его точное местоположение можно определить при помощи Eclipse IDE. Из меню Eclipse **Preferences** (Настройки) перейдите в **Android ▶ Build** (Android ▶ Релиз). Местоположение отладочного сертификата будет указано в поле **Default debug keystore** (Стандартное хранилище отладочных ключей) (рис. 7.5). Если не получается найти меню настройки, то можете прочитать в главе 2, как в него попасть.

Для извлечения «отпечатка» MD5 можно запустить инструмент `keytool` с параметром `-list` (листинг 7.9).

Листинг 7.9. Использование keytool для получения «отпечатка» MD5 отладочного сертификата

```
keytool -list -alias androiddebugkey -keystore
"FULL PATH OF YOUR debug.keystore FILE" -storepass android -keypass android
```

Рис. 7.5. Местоположение отладочного сертификата

Обратите внимание на `alias` отладочного хранилища — `androiddebugkey`. Аналогично пароль к хранилищу ключей будет `android`, пароль к закрытому ключу — тоже `android`. После запуска команды из листинга 7.9 `keytool` предоставит «отпечаток» (рис. 7.6).

Рис. 7.6. Вывод keytool для параметра list (сам «отпечаток» на рисунке замазан)

Теперь вставьте «отпечаток» MD5 вашего сертификата в соответствующее поле на следующем сайте Google: **http://code.google.com/android/maps-api-signup.html**.

Внимательно прочтите условия использования. Если вы согласны с ними, нажмите кнопку **Generate API Key** (Сгенерировать ключ к API), чтобы получить соответствующий ключ от службы Google Maps. Ключ к картографическому API сразу будет готов к применению, с его помощью вы сможете получать от Google картографические данные.

Обратите внимание: для получения такого ключа у вас должен быть аккаунт Google. При генерировании ключа система напомнит вам о необходимости войти в Google под своей учетной записью. Теперь поиграем с картами.

MapView и MapActivity

В значительной мере картографические технологии в Android основаны на элементе управления пользовательским интерфейсом, называемом MapView, и на расширении для android.app.Activity, которое называется MapActivity. Классы MapView и MapActivity решают самые сложные задачи, связанные с отображением карты в Android и управлением ею. Одна из вещей, о которой необходимо помнить, используя эти классы, — то, что они работают вместе. Это означает, что для работы MapView его нужно инстанцировать в рамках MapActivity. Кроме того, при инстанцировании MapView вы сообщаете ключ к картографическому интерфейсу. Если MapView инстанцируется с применением XML-шаблона, вы должны установить свойство android:apiKey. При создании MapView программными методами ключ к картографическому API сообщается конструктору MapView. И наконец, поскольку базовые данные о карте приходят с Google Maps, программе будет нужно право доступа к Интернету. Это означает, что в файле AndroidManifest.xml должен содержаться как минимум следующий запрос:

```
<uses-permission android:name="android.permission.INTERNET" />
```

На самом деле, когда бы вы ни использовали службы, работающие в зависимости от местоположения (карты, GPS и т. д.), добавляйте в файл AndroidManifest.xml три права доступа. Кроме указанного выше, используются еще два права доступа: android.permission.ACCESS_COARSE_LOCATION и android.permission.ACCESS_FINE_LOCATION. В листинге 7.10 полужирным шрифтом выделены записи, которые должны присутствовать в файле AndroidManifest.xml, чтобы картографическое приложение работало.

Листинг 7.10. Теги, которые должны присутствовать в файле AndroidManifest.xml для картографического приложения

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.androidbook"
      android:versionCode="1"
      android:versionName="1.0">
   <application android:icon="@drawable/icon"
           android:label="@string/app_name">
        <uses-library android:name="com.google.android.maps" />
        <activity android:name=".MapViewDemoActivity"
             android:label="@string/app_name">
           <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name=
                   "android.intent.category.LAUNCHER" />
           </intent-filter>
        </activity>
   </application>
   <uses-permission android:name=
        "android.permission.ACCESS_FINE_LOCATION" />
   <uses-permission android:name=
        "android.permission.ACCESS_COARSE_LOCATION" />
```

```
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Как вы помните из табл. 7.2, android.permission.ACCESS_FINE_LOCATION позволяет получать мелкоструктурные данные о местоположении, например GPS. В свою очередь, android.permission.ACCESS_COARSE_LOCATION дает доступ к крупноструктурным данным, в том числе о расположении вышек сотовой связи и точек Wi-Fi.

В файл AndroidManifest.xml нужно внести еще одно изменение. В определении вашего картографического приложения должна содержаться ссылка на библиотеку карт. (Эта строка также присутствует в листинге 7.10.) Пока оставим в стороне необходимые для выполнения условия и рассмотрим рис. 7.7.

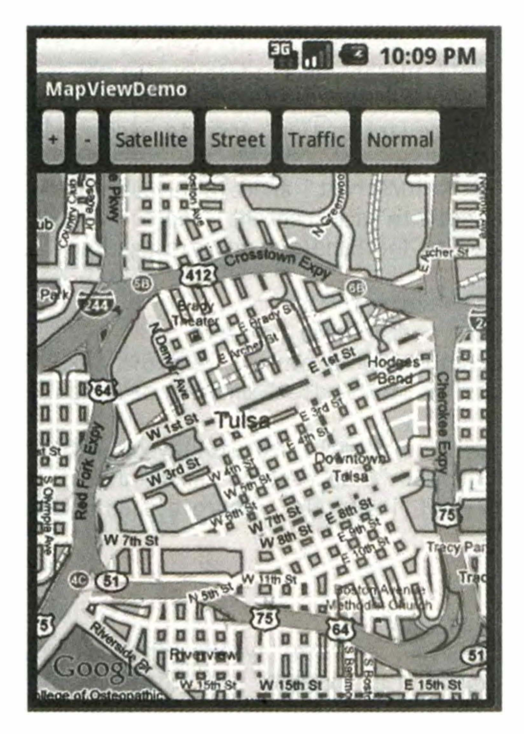


Рис. 7.7. Элемент управления MapView в режиме просмотра улиц

На рис. 7.7 вы видите программу, отображающую карту в режиме просмотра улиц. Программа также позволяет увеличивать или уменьшать фрагмент карты, изменять режим просмотра карты. В листинге 7.11 показан соответствующий XML-шаблон.

Листинг 7.11. XML-шаблон для демонстрационной версии MapView

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/mapview.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent">

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="horizontal" android:layout_width="fill_parent"
   android:layout_height="wrap_content">

   <Button android:id="@+id/zoomin" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="+"
      android:onClick="myClickHandler" android:padding="12px" />

   <Button android:id="@+id/zoomout"
         android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="-"
      android:onClick="myClickHandler" android:padding="12px" />

   <Button android:id="@+id/sat" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Satellite"
      android:onClick="myClickHandler" android:padding="8px" />

   <Button android:id="@+id/street" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Street"
      android:onClick="myClickHandler" android:padding="8px" />

   <Button android:id="@+id/traffic"
         android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Traffic"
      android:onClick="myClickHandler" android:padding="8px" />

   <Button android:id="@+id/normal" android:layout_width="wrap_content"
      android:layout_height="wrap_content" android:text="Normal"
      android:onClick="myClickHandler" android:padding="8px" />

</LinearLayout>

<com.google.android.maps.MapView
   android:id="@+id/mapview" android:layout_width="fill_parent"
   android:layout_height="wrap_content" android:clickable="true"
   android:apiKey=" ЗДЕСЬ НАХОДИТСЯ КЛЮЧ К КАРТОГРАФИЧЕСКОМУ API " />

</LinearLayout>
```

В листинге 7.11 показано, что родительский LinearLayout содержит дочерний LinearLayout и MapView. В дочернем LinearLayout содержатся кнопки, показанные в верхней части рис. 7.7. Не забудьте также обновить значение android:apiKey элемента управления MapView, подставив значение вашего ключа к картографическому API.

Код для нашего примерного картографического приложения показан в листинге 7.12.

Листинг 7.12. Дополнительный класс MapActivity, загружающий XML-шаблон

```
// это файл MapViewDemoActivity.java
import android.os.Bundle;
```

```
import android.view.View;

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;

public class MapViewDemoActivity extends MapActivity
{
   private MapView mapView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.mapview);

      mapView = (MapView)findViewById(R.id.mapview);
   }
   public void myClickHandler(View target) {
      switch(target.getId()) {
      case R.id.zoomin:
         mapView.getController().zoomIn();
         break;
      case R.id.zoomout:
         mapView.getController().zoomOut();
         break;
      case R.id.sat:
         mapView.setSatellite(true);
         break;
      case R.id.street:
         mapView.setStreetView(true);
         break;
      case R.id.traffic:
         mapView.setTraffic(true);
         break;
      case R.id.normal:
         mapView.setSatellite(false);
         mapView.setStreetView(false);
         mapView.setTraffic(false);
         break;
      }
   }

   @Override
   protected boolean isLocationDisplayed() {
      return false;
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }
}
```

Из листинга 7.12 понятно, что отображать MapView при помощи метода onCreate() не сложнее, чем отображать любой другой элемент управления. Это значит, что контент пользовательского интерфейса устанавливается в файле шаблона, содержащем MapView и организующем работу с ним. Работа с функциями увеличения и уменьшения также строится на удивление просто. Для увеличения или уменьшения фрагмента используется класс MapController, относящийся к MapView. Чтобы это осуществить, вызовите mapView.getController(), а затем — соответствующий метод zoomIn() или zoomOut(). Увеличение и уменьшение такого рода является одноуровневым (one-level zoom) — пользователь должен многократно повторять одну и ту же операцию, чтобы увеличить или уменьшить вид.

Не сложнее и процесс изменения режимов просмотра. В MapView поддерживается несколько таких режимов: карта, просмотр улиц, вид со спутника и просмотр транспортного трафика. По умолчанию используется режим карты (Map). При просмотре улиц на карту накладывается слой, на котором голубыми линиями обозначены те улицы, виды которых можно просмотреть. Эти изображения берутся с камер, установленных на машинах, которые проезжают по указанным улицам. Однако обратите внимание: сам элемент MapView не отображает виды улиц. Для просмотра таких изображений нужен специальный элемент управления. Этот вопрос мы подробнее рассмотрим в главе 16. В режиме просмотра со спутника вы видите аэрофотоснимки карты, на которых будут видны крыши зданий, верхушки деревьев, дороги и т. д. В режиме просмотра транспортного трафика информация о загруженности дорог транспортом также представляется в форме цветных линий, обозначающих движущийся транспорт и заторы. Правда, режим просмотра трафика поддерживается только на основных магистралях. Для изменения режима просмотра нужно вызвать соответствующий метод присваивания (setter method) со значением true. В некоторых случаях при установке одного режима выключается другой. Например, невозможно одновременно просматривать карту в режимах просмотра улиц и их загруженности, поэтому при просмотре трафика режим просмотра улиц отключается автоматически. Чтобы отключить режим, присвойте ему значение false.

ПРИМЕЧАНИЕ

Иногда при переходе в режим просмотра улиц или их загруженности ничего не изменяется. Но если сначала установить новый режим, а затем немного переместиться по карте, карта обновится и на ней появится нужная информация.

Чтобы карта двигалась в боковом направлении, установите для MapView в XML атрибут android:clickable="true". Иначе изображение можно будет только увеличивать или уменьшать, но не перемещать по горизонтали. Эту возможность можно также задать в коде, вызвав для MapView метод setClickable(true).

В этом примере есть еще два немаловажных метода — isLocationDisplayed() и isRouteDisplayed(). Они необходимы для выполнения условий пользования службами Google. Программа обязана ответить при помощи true или false, чтобы сообщить картографическому серверу, должно ли отображаться актуальное местоположение устройства, а также должны ли отображаться какие-либо сведения о маршруте, в частности направления движения.

Согласитесь, объем кода, требуемый в Android для отображения карты, реализации функций уменьшения и увеличения масштаба и смены режимов просмотра, минимален (см. листинг 7.12). Тем не менее есть еще более простой способ внедрения в программу элементов управления, предназначенных для фокусировки. Рассмотрим XML-шаблон и код из листинга 7.13.

Листинг 7.13. Фокусировка упрощается

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/mapview.xml -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="fill_parent">

   <com.google.android.maps.MapView android:id="@+id/mapview"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:clickable="true"
      android:apiKey="ЗДЕСЬ НАХОДИТСЯ КЛЮЧ К КАРТОГРАФИЧЕСКОМУ API"
      />
</RelativeLayout>

public class MapViewDemoActivity extends MapActivity
{
   private MapView mapView;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.mapview);

      mapView = (MapView)findViewById(R.id.mapview);
      mapView.setBuiltInZoomControls(true);
   }

   @Override
   protected boolean isLocationDisplayed() {
      return false;
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }
}
```

Разница между листингами 7.12 и 7.13 состоит в том, что мы изменили XML-шаблон нашего вида и перешли к использованию `RelativeLayout`. Мы также удалили все элементы управления, обеспечивающие фокусировку и работу с режимами просмотра. Вся прелесть этого примера в коде, а не в шаблоне. В `MapView` уже есть элементы, предназначенные для фокусировки. Все, что нужно сделать, — включить

их при помощи метода setBuiltInZoomControls(). На рис. 7.8 показаны элементы фокусировки, по умолчанию присутствующие в MapView.

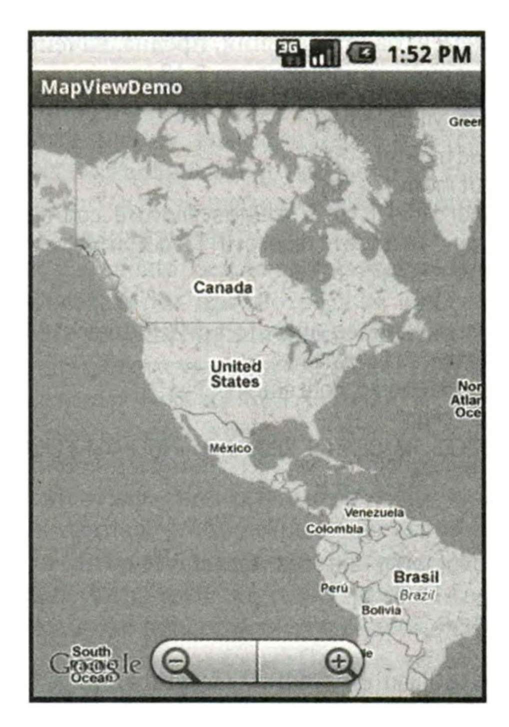


Рис. 7.8. Интегрированные в MapView элементы для работы с фокусировкой

Теперь узнаем, как добавлять на карту пользовательские данные.

Использование наложений

В Google Maps есть функция, позволяющая пользователю добавлять собственные данные поверх карты. Чтобы опробовать ее на практике, попробуйте поискать пиццерию где-нибудь поблизости от своего дома. Google Maps разместит над каждой пиццерией «иголочки» — шарики-маркеры. Эта возможность в Google Maps реализуется путем добавления поверх карты дополнительного слоя. Основным классом для обеспечения такой функциональности является Overlay, но можно использовать и расширение этого класса ItemizedOverlay. В листинге 7.14 показан соответствующий пример.

Листинг 7.14. Нанесение на карту дополнительных отметок при помощи ItemizedOverlay

```
import java.util.ArrayList;
import java.util.List;

import android.graphics.Canvas;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
```

```
import android.widget.LinearLayout;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.OverlayItem;

public class MappingOverlayActivity extends MapActivity {
   private MapView mapView;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.mapview);

      mapView = (MapView) findViewById(R.id.mapview);

      mapView.setBuiltInZoomControls(true);

      mapView.setClickable(true);

      Drawable marker=getResources().getDrawable(R.drawable.mapmarker);
      marker.setBounds(0, 0, marker.getIntrinsicWidth(),
                       marker.getIntrinsicHeight());

      InterestingLocations funPlaces = new InterestingLocations(marker);
      mapView.getOverlays().add(funPlaces);

      GeoPoint pt = funPlaces.getCenter();  // получение точки
                               // с наиболее высоким рейтингом
      mapView.getController().setCenter(pt);
      mapView.getController().setZoom(15);
   }

   @Override
   protected boolean isLocationDisplayed() {
      return false;
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }

   class InterestingLocations extends ItemizedOverlay {
      private List<OverlayItem> locations = new ArrayList<OverlayItem>();
      private Drawable marker;
```

```
    public InterestingLocations(Drawable marker)
    {
       super(marker);
       this.marker=marker;
       // создание интересующих нас мест
       GeoPoint disneyMagicKingdom = new
  GeoPoint((int)(28.418971*1000000),(int)(-81.581436*1000000));
       GeoPoint disneySevenLagoon = new
  GeoPoint((int)(28.410067*1000000),(int)(-81.583699*1000000));

       locations.add(new OverlayItem(disneyMagicKingdom ,
                   "Magic Kingdom", "Magic Kingdom"));
       locations.add(new OverlayItem(disneySevenLagoon ,
                   "Seven Lagoon", "Seven Lagoon"));

       populate();
    }

    @Override
    public void draw(Canvas canvas, MapView mapView, boolean shadow) {
       super.draw(canvas, mapView, shadow);

       boundCenterBottom(marker);
    }

    @Override
    protected OverlayItem createItem(int i) {
       return locations.get(i);
    }

    @Override
    public int size() {
       return locations.size();
    }
  }
}
```

В листинге 7.14 показано, как маркеры накладываются на карту. В этом примере использованы два маркера: один указывает на «Волшебное Королевство» Диснея, а другой — на диснеевскую «Лагуну семи морей» (это знаменитые аттракционы, оба находятся вблизи города Орландо, штат Флорида, США; рис. 7.9).

ПРИМЕЧАНИЕ

Чтобы этот пример работал, у вас должен быть отрисовываемый объект, который будет выполнять на карте роль маркера. Это изображение должно быть сохранено в каталоге /res/drawable, чтобы ссылка на ID ресурса, используемая при вызове getDrawable(), подходила для имени, которое вы выбрали для вашего файла. При наложении для маркера необходимо задать точку привязки (anchor point), то есть интересующую вас точку на карте, на которую и будет указывать маркер. Например, в рамках метода draw() мы вызываем boundCenterBottom(). Таким образом, точка привязки маркера будет располагаться с его нижнего края. Другой метод задания точки привязки — boundCenter(). Он располагает центр маркера точно над той точкой карты, куда нужно указать.

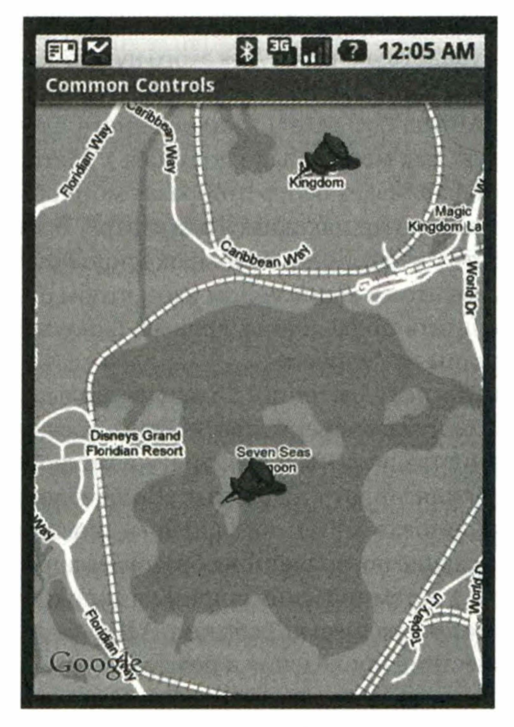


Рис. 7.9. MapView с маркерами

Чтобы на карту можно было добавлять маркеры, нужно создать и внедрить расширение com.google.android.maps.Overlay. Сам класс Overlay в данном случае инстанцировать нельзя, поэтому его потребуется расширить либо использовать одно из имеющихся расширений. В нашем примере мы реализовали InterestingLocations, дополняющий ItemizedOverlay, а он, в свою очередь, дополняет Overlay. Класс Overlay определяет степень сжатия (contract) наложенного слоя, а ItemizedOverlay очень удобен при создании списка мест, которые можно отметить на карте.

Обычно работа строится так. Класс ItemizedOverlay расширяется, и к нему добавляются «элементы» — интересующие нас места, расположенные на карте. Это делается при помощи конструктора. После инстанцирования интересующих нас точек вызывается метод populate() класса ItemizedOverlay. Метод populate() кэширует OverlayItem(s). Внутри системы класс вызывает метод size(), предназначенный для определения количества элементов в накладываемом слое, а затем запускает цикл, в котором вызывает createItem(i) для каждого элемента. Метод createItem позволяет возвратить готовый элемент при наличии индекса, под которым этот элемент расположен в массиве.

Из листинга 7.14 понятно, что можно просто создать точки и вызвать populate(), чтобы маркеры отобразились на карте. Остальная работа выполняется при помощи сжатия Overlay. Чтобы все это работало, метод onCreate() явления создает экземпляр InterestingLocations, передаваемый в Drawable, который используется в качестве маркера. Затем onCreate() добавляет экземпляр InterestingLocations к коллекции накладываемых слоев (mapView.getOverlays().add()).

Теперь мы ассоциировали с картой слой, но нам еще нужно переместиться по карте на экране, чтобы увидеть маркеры. Для этого нужно совместить точку и центр отображаемого фрагмента карты. Используем первую точку наложенного слоя в качестве центра карты. Метод getCenter() наложенного слоя возвращает первую точку (а не центральную, как можно было подумать). Метод setCenter() элемента управления mapview устанавливает центр отображаемого объекта. Метод setZoom() определяет, насколько высоко мы находимся над картой. В этом демонстрационном примере мы для удобства выбрали уровень фокусировки 15. Нам следовало бы провести итерацию элементов в наложенном слое, чтобы определить его внешние границы, а затем рассчитать подходящий уровень фокусировки, чтобы на карте были одновременно видны все маркеры.

Еще один интересный аспект листинга 7.15 заключается в создании OverlayItem(s). Для создания OverlayItem нужен объект типа GeoPoint. Класс GeoPoint представляет точку на карте в координатах широты и долготы — в микроградусах. В нашем примере мы получили значения широты и долготы «Волшебного Королевства» и «Лагуны семи морей», использовав сайты, посвященные геокодированию. (Вскоре мы покажем, как геокодирование позволяет преобразовать адрес, например, в координаты широты и долготы.) Затем значения широты и долготы переводятся в микроградусы — именно с микроградусами работает API — выполнив умножение на 1 000 000, а затем осуществив приведение к целому.

Мы показали, как размещать маркеры на карте. Однако при наложении можно не только расставлять указатели, но и выполнять многие другие операции. Например, при перемещении по карте можно отображать анимацию различных товаров, либо показывать погоду, допустим символы атмосферных фронтов и гроз.

В любом случае вы, должно быть, согласитесь, что процесс установки маркеров на карту проще быть не может. Или все-таки может? У нас нет базы данных, в которой хранились бы парные значения широты, но мы все же понимаем, что нам потребуется создать один или несколько объектов GeoPoint на базе реальных адресов. Именно для этого нам пригодится Geocoder, входящий в состав пакета для работы с местоположением. В следующем подразделе мы рассмотрим использование Geocoder.

Пакет Location

В пакете android.location содержатся функции для эксплуатации служб, работа которых зависит от местоположения устройства. В этом разделе мы обсудим два важных фрагмента этого пакета — класс Geocoder и службу LocationManager. Начнем с Geocoder.

Геокодирование в Android

Если вы собираетесь решать при помощи карт какие-либо практические задачи, то вам, вероятно, не раз придется преобразовывать адрес в парные значения широты и долготы. Такая операция называется *геокодированием*. В Android эту функцию

обеспечивает класс android.location.Geocoder. На самом деле этот класс обеспечивает как прямые, так и обратные преобразования — он может брать адрес и возвращать пару «широта/долгота», а также переводить пары «широта/долгота» в списки адресов. В классе присутствуют следующие методы:

- List<Address> getFromLocation(double latitude, double longitude, int maxResults);
- List<Address> getFromLocationName(String locationName, int maxResults, double lowerLeftLatitude, double lowerLeftLongitude, double upperRightLatitude, double upperRightLongitude);
- List<Address> getFromLocationName(String locationName, int maxResults).

Оказывается, что расчет адреса совсем не точная наука, так как местоположение может быть описано различными способами. Например, методы getFromLocationName() могут принимать название места, физический адрес, код аэропорта либо просто наиболее известное обозначение конкретного места. Следовательно, методы предоставляют не одиночный адрес, а список адресов. Поскольку метод возвращает список, целесообразно ограничить количество результатов в наборе, определив значение maxResults в пределах от 1 до 5. Рассмотрим пример.

В листинге 7.15 приведен XML-файл шаблона и соответствующий код для пользовательского интерфейса, показанного на рис. 7.10. Чтобы запустить пример, вставьте в листинг ваш ключ к картографическому API.

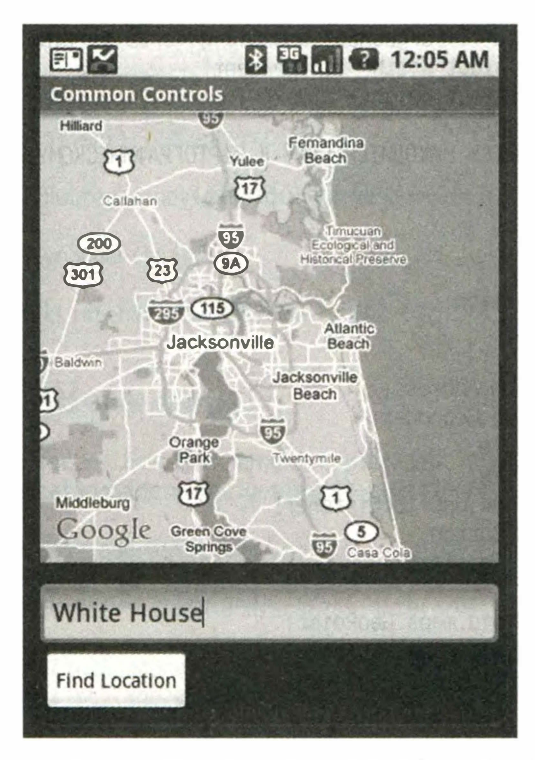


Рис. 7.10. Геокодирование точки на базе имеющегося названия места

Листинг 7.15. Работа с классом Geocoder в Android

```
<?xml version="1.0" encoding="utf-8"?>
<!--Это файл /res/layout/geocode.xml -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

      <LinearLayout android:layout_width="fill_parent"
         android:layout_alignParentBottom="true"
         android:layout_height="wrap_content"
         android:orientation="vertical" >

         <EditText android:layout_width="fill_parent"
            android:id="@+id/location"
            android:layout_height="wrap_content"
            android:text="White House"/>

         <Button android:id="@+id/geocodeBtn"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Find Location"/>
      </LinearLayout>

      <com.google.android.maps.MapView
            android:id="@+id/geoMap" android:clickable="true"
            android:layout_width="fill_parent"
            android:layout_height="320px"
            android:apiKey=
                  "ЗДЕСЬ НАХОДИТСЯ КЛЮЧ К КАРТОГРАФИЧЕСКОМУ API"
            />

</RelativeLayout>

import java.io.IOException;
import java.util.List;

import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;

public class GeocodingDemoActivity extends MapActivity
{
```

```
Geocoder geocoder = null;
MapView mapView = null;

@Override
protected boolean isLocationDisplayed() {
   return false;
}

@Override
protected boolean isRouteDisplayed() {
   return false;
}

@Override
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);

   setContentView(R.layout.geocode);
   mapView = (MapView)findViewById(R.id.geoMap);
   mapView.setBuiltInZoomControls(true);

   // широта/долгота города Джэксонвилл, штат Флорида, США
   int lat = (int)(30.334954*1000000);
   int lng = (int)(-81.5625*1000000);
   GeoPoint pt = new GeoPoint(lat,lng);
   mapView.getController().setZoom(10);
   mapView.getController().setCenter(pt);

   Button geoBtn =(Button)findViewById(R.id.geocodeBtn);

   geocoder = new Geocoder(this);

   geoBtn.setOnClickListener(new OnClickListener(){

   @Override
   public void onClick(View arg0) {
      try {
         EditText loc = (EditText)findViewById(R.id.location);
         String locationName = loc.getText().toString();

         List<Address> addressList =
               geocoder.getFromLocationName(locationName, 5);
         if(addressList!=null && addressList.size()>0)
         {
         int lat = (int)(addressList.get(0).getLatitude()*1000000);
         int lng = (int)(addressList.get(0).getLongitude()*1000000);

         GeoPoint pt = new GeoPoint(lat,lng);
         mapView.getController().setZoom(15);
```

```
                mapView.getController().setCenter(pt);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }});
    }
}
```

Чтобы опробовать геокодирование в Android на практике, укажите название точки либо ее адрес в поле EditText, а затем нажмите кнопку **Find Location** (Найти местоположение). Чтобы найти адрес места, вызовите метод getFromLocationName() класса Geocoder. Место на карте может иметь общеизвестное название, например White House (Белый дом). На геокодирование может уйти определенное время, поэтому мы рекомендуем ограничивать количество ресурсов до пяти — о том же говорится и в документации по Android. При вызове метода getFromLocationName() вы получаете список адресов. Демоприложение берет список адресов и обрабатывает первый — если адреса были найдены. Каждый адрес имеет значения широты и долготы, используемые для создания GeoPoint. Затем вы получаете инструмент управления картой и с его помощью переходите к точке. Для степени фокусировки можно задать значение от 1 до 21 включительно. При каждом переходе между этими значениями коэффициент фокусировки умножается на 2.

Важно понимать несколько аспектов, связанных с геокодированием. Во-первых, возвращенный адрес не всегда точен. Разумеется, итоговый список адресов зависит от степени точности ввода, поэтому всячески старайтесь сообщить Geocoder точное название места. Во-вторых, старайтесь устанавливать значение для maxResults в диапазоне от 1 до 5. Наконец, очень рекомендуем выполнять операцию геокодирования и управление пользовательским интерфейсом в разных потоках. На это есть две причины. Первая очевидна: на геокодирование уходит немало времени, и вам совсем не захочется, чтобы на это время пользовательский интерфейс зависал. Вторая причина заключается в том, что на мобильном устройстве нужно всегда быть готовым к внезапному разрыву соединения с сетью либо к тому, что такое соединение будет слабым. Следовательно, исключения ввода-вывода и задержки (timeouts) требуют соответствующей обработки. Рассчитав адреса, вы сможете послать результаты в поток, работающий с пользовательским интерфейсом. Рассмотрим этот вопрос подробнее.

Геокодирование с применением фоновых потоков

При выполнении операций, на которые тратится много времени, обычно применяются фоновые потоки (background threads). Чаще всего работа строится так: обрабатывается событие, происходящее в пользовательском интерфейсе (например, нажатие кнопки), и при этом инициируется долговременная операция. От обработчика событий создается новый поток, в котором выполняется работа, и потом этот новый поток запускается. После этого поток UI возвращается к работе с пользовательским интерфейсом, в частности продолжает взаимодействовать с пользователем, а фоновый поток в это время продолжает работать. После завершения

работы фонового потока, возможно, потребуется обновить часть пользовательского интерфейса либо пользователь получит уведомление. Фоновый поток не обновляет пользовательский интерфейс напрямую; вместо этого он посылает потоку пользовательского интерфейса сообщение о необходимости обновиться. В листинге 7.16 эта идея проиллюстрирована применительно к геокодированию. Мы воспользуемся тем же файлом geocode.xml, что и раньше. Кроме того, можно применить прежний файл AndroidManifest.xml.

Листинг 7.16. Геокодирование в отдельном потоке

```
import java.io.IOException;
import java.util.List;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.ProgressDialog;
import android.location.Address;
import android.location.Geocoder;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
public class GeocodingDemoActivity extends MapActivity
{
   Geocoder geocoder = null;
   MapView mapView = null;
   ProgressDialog progDialog=null;
   List<Address> addressList=null;

   @Override
   protected boolean isLocationDisplayed() {
      return false;
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.geocode);
```

```
    mapView = (MapView)findViewById(R.id.geoMap);
    mapView.setBuiltInZoomControls(true);

    // широта/долгота города Джэксонвилл, штат Флорида, США
    int lat = (int)(30.334954*1000000);
    int lng = (int)(-81.5625*1000000);
    GeoPoint pt = new GeoPoint(lat,lng);
    mapView.getController().setZoom(10);
    mapView.getController().setCenter(pt);

    Button geoBtn =(Button)findViewById(R.id.geocodeBtn);

    geocoder = new Geocoder(this);

    geoBtn.setOnClickListener(new OnClickListener(){

       @Override
       public void onClick(View view) {
          EditText loc = (EditText)findViewById(R.id.location);
          String locationName = loc.getText().toString();

          progDialog =
             ProgressDialog.show(GeocodingDemoActivity.this,
             "Processing...", "Finding Location...", true, false);
          findLocation(locationName);
       }});
}
private void findLocation(final String locationName)
{
   Thread thrd = new Thread()
   {
      public void run()
      {
         try {
            // выполнение фоновой работы
            addressList = geocoder.getFromLocationName
                        (locationName, 5);
            // отправка сообщения обработчику о необходимости
            // обработать результат процесса
            uiCallback.sendEmptyMessage(0);

         } catch (IOException e) {
            e.printStackTrace();
         }
      }
   };
   thrd.start();
}
// обработчик обратных вызовов потока пользовательского интерфейса
```

```
    private Handler uiCallback = new Handler()
    {
        @Override
        public void handleMessage(Message msg)
        {
            progDialog.dismiss();

            if(addressList!=null && addressList.size()>0)
            {
                int lat = (int)(addressList.get(0).getLatitude()*1000000);
                int lng = (int)(addressList.get(0).getLongitude()*1000000);
                GeoPoint pt = new GeoPoint(lat,lng);
                mapView.getController().setZoom(15);
                mapView.getController().setCenter(pt);
            }
            else
            {
                Dialog foundNothingDlg = new
                        AlertDialog.Builder(GeocodingDemoActivity.this)
                    .setIcon(0)
                    .setTitle("Failed to Find Location")
                    .setPositiveButton("Ok", null)
                    .setMessage("Location Not Found...")
                .create();
                foundNothingDlg.show();
            }
        }
    };
}
```

В листинге 7.16 показана модифицированная версия кода из листинга 7.15. Разница в том, что теперь метод onClick() отображает диалоговое окно хода выполнения программы (progress dialog) и вызывает findLocation() (рис. 7.11). Затем findLocation() создает новый поток и вызывает метод start(), в результате которого метод run() наконец вызывает выполнение потока. В методе run() используется класс Geocoder, предназначенный для поиска места на карте. После завершения поиска нужно послать сообщение определенному элементу, который умеет устанавливать связь с потоком пользовательского интерфейса, так как карту нужно обновить. Для этого в Android применяется класс android.os.Handler. От фонового потока вызывается uiCallback.sendEmptyMessage(0) — метод обратного вызова, предназначенный для получения от потока пользовательского интерфейса результата обработки данных поиска. В нашем случае в сообщении не нужно посылать никакой информации, так как данные совместно применяются через addressList. Код обращается к обратному вызову обработчика событий, который сворачивает диалоговое окно, а затем просматривает addressList, возвращенный Geocoder. После этого обратный вызов обновляет карту полученным результатом либо выводит окно с предупреждением, указывающее, что поиск результатов не дал. Соответствующий пользовательский интерфейс показан на рис. 7.11.

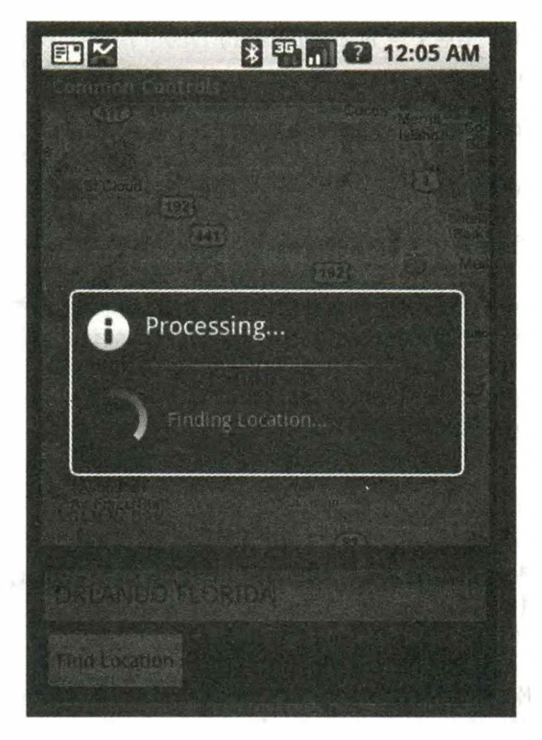


Рис. 7.11. Показ окна хода выполнения программы, если операция является длительной

Служба LocationManager

Служба LocationManager является одной из ключевых составляющих пакета android.location. Она обеспечивает выполнение двух задач: предоставляет механизм для получения информации о географическом расположении устройства и возможность уведомления (при помощи намерения) о том, что устройство оказалось в указанном районе местности.

В этом подразделе мы изучим, как работает служба LocationManager. Чтобы ее использовать, нужно сначала получить ссылку на нее. В листинге 7.17 показан принцип работы со службой LocationManager.

Листинг 7.17. Работа со службой LocationManager

```
import java.util.List;

import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;

public class LocationManagerDemoActivity extends Activity
{

   @Override
```

```
    protected void onCreate(Bundle savedInstanceState)
    {

        super.onCreate(savedInstanceState);
        LocationManager locMgr =(LocationManager)this.getSystemService
                        (Context.LOCATION_SERVICE);
        Location loc =
              locMgr.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        List<String> providerList = locMgr.getAllProviders();

    }
}
```

LocationManager — это системная служба (system-level service). Системными называются службы, которые вы получаете из контекста, используя имя службы. Их непосредственного инстанцирования не происходит. В классе android.app.Activity имеется служебный метод getSystemService(), который можно использовать для получения системных служб. Как показано в листинге 7.17, можно вызвать getSystemService() и передать с ним название службы, которая вам нужна (в данном случае Context.LOCATION_SERVICE).

Служба LocationManager представляет подробные географические данные, используя при этом поставщик местоположения (location provider). В настоящее время существует два типа поставщиков местоположения: GPS и сетевые. GPS предоставляет доступ к глобальной системе определения местоположения для получения данных о местонахождении устройства; в свою очередь, сеть позволяет использовать для этой цели сотовые вышки или Wi-Fi. Класс LocationManager может сообщить последнее известное место, в котором находилось устройство (которое должно находиться недалеко оттуда, где устройство располагается сейчас), при помощи метода getLastKnownLocation(). Информация о местоположении передается поставщиком, то есть метод принимает в качестве одного из параметров имя поставщика местоположения, который вы хотите использовать. Имена поставщиков местоположения строятся по принципу LocationManager.GPS_PROVIDER и LocationManager.NETWORK_PROVIDER. При вызове getLastKnownLocation() возвращается экземпляр android.location.Location. Класс Location дает координаты широты и долготы, по которым находится устройство, представляет информацию о времени, когда система в последний раз вычислила местоположение устройства, а также, возможно, отображает высоту, на которой находится устройство, скорость его перемещения и направление по компасу.

Поскольку LocationManager работает с поставщиками, в классе имеется специальный прикладной программный интерфейс для получения поставщиков. Например, можно получить всех поставщиков, вызвав getAllProviders(). Конкретный поставщик можно получить, вызвав getProvider() и передав ему имя поставщика в качестве аргумента (например, LocationManager.GPS_PROVIDER).

В таких случаях на этапе разработки службы LocationManager вызывают сбой в программе — LocationManager требует информацию о местоположении, а эмулятор не имеет доступа ни к GPS, ни к сотовым вышкам. Итак, чтобы приступить к разработке LocationManager, вы (как бы) сообщаете эмулятору свое местоположение. Например, можно запросить у LocationManager соответствующее уведомление, если устройство находится вблизи от вас. Чтобы протестировать в эмуляторе подобный

случай, можно посылать эмулятору периодические уведомления о вашем местоположении. Эмулятор будет посылать эту информацию обратно в программу. Соответствующий пример показан в листинге 7.18.

Листинг 7.18. Регистрация для получения данных об изменении местоположения

```
import android.app.Activity;
import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.Toast;

public class LocationUpdateDemoActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);

      LocationManager locMgr = (LocationManager)
               getSystemService(Context.LOCATION_SERVICE);
      LocationListener locListener = new LocationListener()
      {

         public void onLocationChanged(Location location)
         {
            if (location != null)
            {
               Toast.makeText(getBaseContext(),
                  "New location latitude [" +
                     location.getLatitude() +
                     "] longitude [" + location.getLongitude()+"]",
                     Toast.LENGTH_SHORT).show();
            }
         }

         public void onProviderDisabled(String provider)
         {
         }

         public void onProviderEnabled(String provider)
         {
         }

         public void onStatusChanged(String provider,
                              int status, Bundle extras)
         {
         }
      };

      locMgr.requestLocationUpdates(
```

```
            LocationManager.GPS_PROVIDER,
            0,    // минимальное время в миллисекундах
            0,    // минимальное расстояние в метрах
            locListener);
    }
}
```

Как было указано выше, одним из основных видов практического применения службы LocationManager является получение данных о местонахождении устройства. В листинге 7.18 показано, как зарегистрировать слушателя, который будет обрабатывать события, связанные с изменением местоположения. Для регистрации слушателя вызывается метод requestLocationUpdates(), ему передается в качестве одного из параметров тип поставщика. Если местонахождение устройства изменится, вызывается метод onLocationChanged() слушателя с новым значением Location. В нашем примере минимальные значения времени и расстояния (minTime и minDistance) равны 0. Это означает, что LocationManager должен присылать нам обновления сразу же по мере их поступления. На практике такие настройки применяются редко, но здесь мы используем именно их, чтобы демоверсия работала быстрее. (На практике вам не понадобится, чтобы оборудование определяло свое актуальное положение так часто — из-за этого будет быстро разряжаться батарея.) Эти значения устанавливаются исходя из ситуации и сводят количество уведомлений об изменении положения устройства к необходимому минимуму.

В листинге 7.18 вы познакомились с новым инструментом: виджетом Toast. Это удобный элемент, позволяющий показывать пользователю маленькие всплывающие сообщения, в которых кратко описываются происходящие изменения. Такое сообщение появляется при проведении указателем над имеющимся видом, а потом само исчезает. Длительность показа такого сообщения регулируется при помощи LENGTH_LONG вместо LENGTH_SHORT.

Чтобы протестировать эту возможность в эмуляторе, воспользуйтесь интерфейсом DDMS (Dalvik Debug Monitor Service), входящим в состав ADT-плагина для Eclipse. Пользовательский интерфейс DDMS имеет специальное окно, через которое можно сообщить эмулятору об изменении местонахождения (рис. 7.12).

Из рис. 7.12 видно, что вкладка **Manual** (Вручную) пользовательского интерфейса DDMS позволяет отправлять в эмулятор новые данные о положении устройства в системе GPS (пара «широта/долгота»). При посылке новых данных о местоположении запускается метод слушателя onLocationChanged(), выдающий пользователю сообщение о перемещении на новое место.

Данные о местоположении можно послать на эмулятор и некоторыми другими способами. Они показаны в интерфейсе DDMS (см. рис. 7.12). Например, интерфейс DDMS позволяет отправлять файлы в формате обмена данными GPS (GPS Exchange Format, GPX) или в формате языка разметки Keyhole (KML). Образцы файлов в формате GPX можно взять на следующих сайтах:

- http://www.topografix.com/gpx_resources.asp;
- http://tramper.co.nz/?view=gpxFiles;
- http://www.gpxchange.com/.

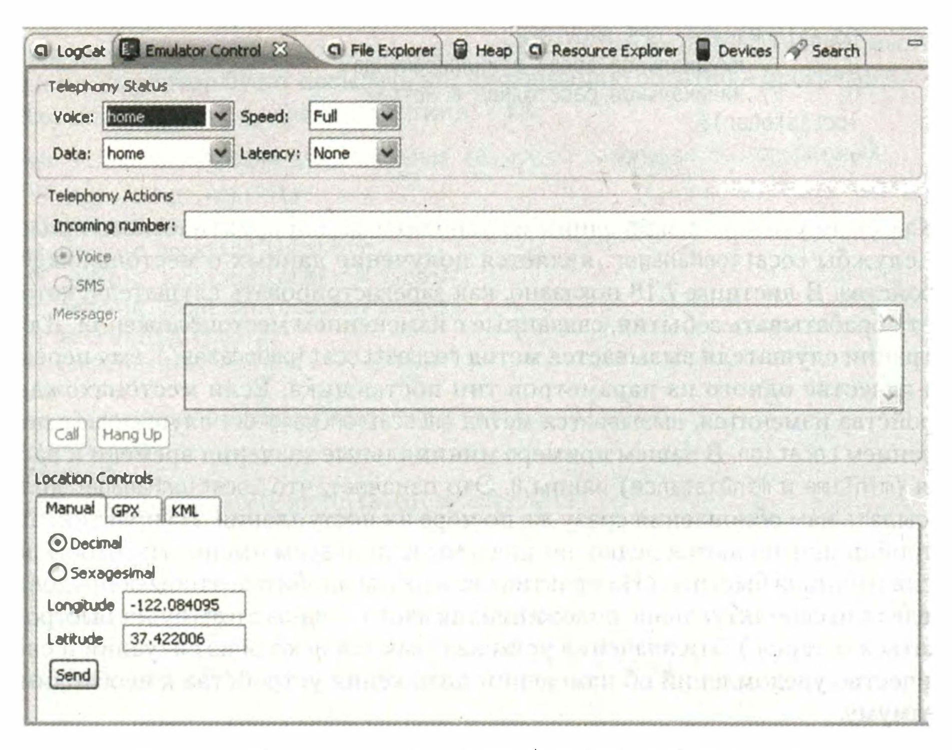


Рис. 7.12. Работа с пользовательским интерфейсом DDMS в Eclipse; сообщение эмулятору данных о местоположении устройства

Для создания и получения образцов файлов в формате KML обратитесь к следующим ресурсам:

- http://bbs.keyhole.com/;
- http://code.google.com/apis/kml/documentation/kml_tut.html.

ПРИМЕЧАНИЕ

На некоторых сайтах размещаются файлы в формате KMZ. Это файлы KML в ZIP-архиве, то есть для получения файла KML их нужно просто разархивировать. Некоторые файлы KML содержат значения из пространства имен XML; их нужно изменить, чтобы такие файлы правильно работали в DDMS. Если у вас возникают проблемы с каким-либо XML-файлом, убедитесь, что в нем есть строка <kml xmlns="http://earth.google.com/kml/2.x">.

Можно закачать файл GPX или KML в эмулятор и установить скорость, с которой файл будет воспроизводиться в эмуляторе (рис. 7.13). После этого эмулятор будет посылать программе информацию об изменении местоположения с указанной скоростью. На рис. 7.13 видно, что в GPX-файле есть точки (points), находящиеся в верхней части, и пути (paths), показываемые в нижней части. Воспроизвести точку нельзя, но если щелкнуть на ней, точка будет послана в эмулятор. Можно щелкнуть на пути, а потом на кнопке Play (Воспроизвести) — кнопка включится, и после этого можно будет воспроизводить точки.

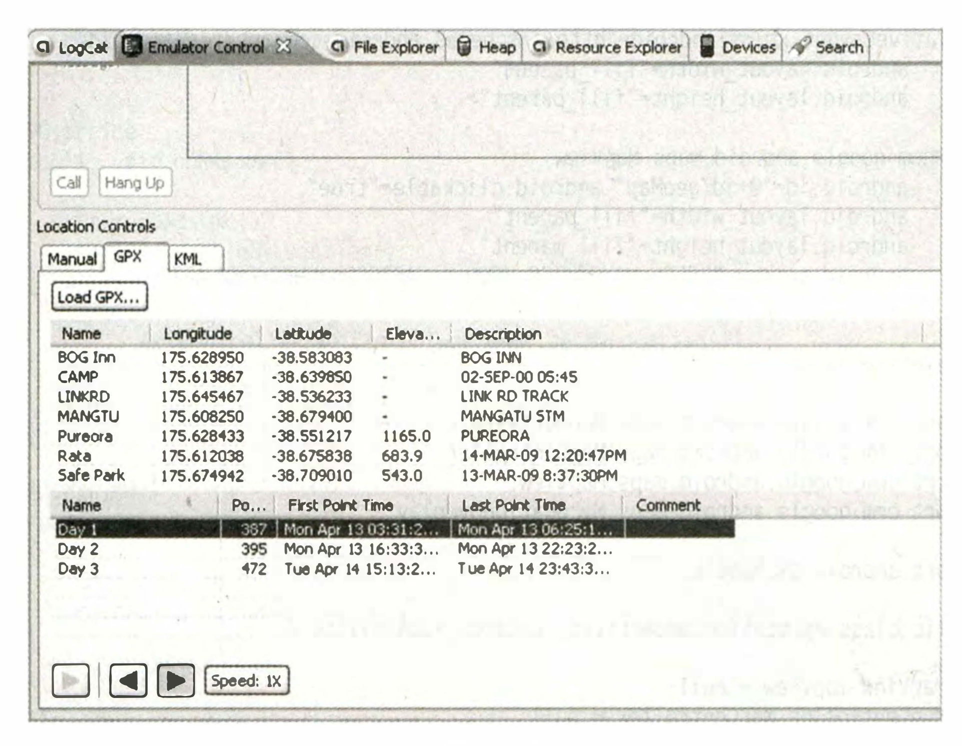


Рис. 7.13. Загрузка в эмулятор файлов GPX и KML для последующего воспроизведения

ПРИМЕЧАНИЕ

Имеются сообщения о том, что не все GPX-файлы могут быть интерпретированы в Emulator Control. Если вы пытаетесь загрузить GPX-файл и ничего не происходит, попробуйте другой файл из другого источника.

Использование MyLocationOverlay

Обычно GPS и карты применяются для того, чтобы указать пользователю, где он находится. В Android эта задача решается очень просто — путем наложения специального слоя, называемого `MyLocationOverlay`. Добавив такой слой в `MapView`, вы можете поместить в нем мерцающую голубую точку, которая будет перемещаться по карте вместе с вами, отражая, таким образом, информацию, поступающую от службы `LocationManager`.

В следующем примере мы объединим несколько концепций в одно приложение. Создайте новый проект Android и назовите его `MyLocationDemoActivity`. Далее выберите функцию **Build Target** (Построить цель) Google API. При помощи листинга 7.19 обновите файлы `main.xml` и `MyLocationDemoActivity.java`.

Листинг 7.19. Работа с MyLocationOverlay

```
<?xml version="1.0" encoding="utf-8"?>
<!--Это файл /res/layout/main.xml -->
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">

   <com.google.android.maps.MapView
      android:id="@+id/geoMap" android:clickable="true"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:apiKey="ЗДЕСЬ НАХОДИТСЯ КЛЮЧ К КАРТОГРАФИЧЕСКОМУ API"
      />

</RelativeLayout>

import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;

import android.os.Bundle;

public class MyLocationDemoActivity extends MapActivity {

   MapView mapView = null;
   MapController mapController = null;
   MyLocationOverlay whereAmI = null;

   @Override
   protected boolean isLocationDisplayed() {
      return whereAmI.isMyLocationEnabled();
   }

   @Override
   protected boolean isRouteDisplayed() {
      return false;
   }

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      mapView = (MapView)findViewById(R.id.geoMap);
      mapView.setBuiltInZoomControls(true);

      mapController = mapView.getController();
      mapController.setZoom(15);

      whereAmI = new MyLocationOverlay(this, mapView);
      mapView.getOverlays().add(whereAmI);
```

```
        mapView.postInvalidate();
    }

    @Override
    public void onResume()
    {
        super.onResume();
        whereAmI.enableMyLocation();
        whereAmI.runOnFirstFix(new Runnable() {
            public void run() {
                mapController.setCenter(whereAmI.getMyLocation());
            }
        });
    }

    @Override
    public void onPause()
    {
        super.onPause();
        whereAmI.disableMyLocation();
    }
}
```

Не забудьте изменить суперкласс с Activity на MapActivity и включить соответствующий импорт. Потребуется также обновить файл AndroidManifest.xml — включить в него соответствующие теги uses-permission и uses-library, о которых мы говорили выше. Обратите внимание: в этом примере метод isLocationDisplayed() вернет true, если в настоящий момент мы показываем актуальное расположение устройства на карте.

После запуска такого приложения в эмуляторе нужно начать посылать программе обновления информации о местоположении, а потом вы удивитесь тому, что произойдет. Для этого перейдите к виду **DDMS Emulator Control** в Eclipse — как это сделать, мы рассказали выше в этом подразделе. В Интернете нужно найти образец GPX-файла. Таких файлов много на сайтах, список которых также был приведен чуть выше. Просто выберите один из файлов и скачайте его на свою рабочую станцию. Затем загрузите файл в **Emulator Control** кнопкой **Load GPX** (загрузить GPX) на вкладке **GPX** под **Location Controls** (Элементы управления местоположением). Выберите из нижнего списка путь и нажмите кнопку **Play** (Воспроизведение), которая обозначена зеленой стрелкой. Обратите также внимание на кнопку **Speed** (Скорость). Так вы запускаете поток обновлений данных о местоположении в эмулятор, а ваша программа будет их принимать. Чтобы обновления происходили чаще, нажмите кнопку **Speed** (Скорость).

Код, приведенный выше, совсем незамысловат. После настройки базовых данных MapView, подготовки элементов управления для работы с фокусировкой мы создадим наложение MyLocationOverlay. Наложим новый слой на **MapView**, затем применим к нему метод postInvalidate() — и новый слой появится на экране. Без последнего вызова слой будет создан, но не будет отображаться.

Помните, что ваша программа будет вызывать onResume(), даже если вы только что ее включили, а также после выхода из спящего режима. Инициируем отслеживание местоположения (location tracking) при помощи onResume(), а отключаем при помощи onPause(). Нет смысла истощать батарею запросами о местоположении, если мы не собираемся ими пользоваться. Но нам может понадобиться не только активировать onResume(), но и перейти к тому месту на карте, где мы сейчас находимся. В классе MyLocationOverlay есть полезный метод для этой цели — runOnFirstFix(). Он позволяет установить код, который будет выполняться сразу же после получения данных о местоположении. Это может произойти немедленно, так как у нас уже есть данные о нашем последнем местоположении, или позже — после получения новой информации от GPS_PROVIDER или NETWORK_PROVIDER. Итак, имея точку привязки, мы располагаем ее в центре карты. На этом наша работа завершается, поскольку сам MyLocationOverlay получает обновления данных о местоположении и обозначает изменения нашего местонахождения мерцающей голубой точкой.

Обратите внимание: вы можете работать с фокусировкой в процессе получения обновлений. Вы даже можете перейти по карте в сторону от точки вашего текущего местонахождения. Сложно сказать, хорошо это или плохо. Если вы отойдете по карте от точки, в которой были, и забудете, куда нужно вернуться, вам будет сложно найти себя без того, чтобы уменьшить масштаб и отыскать голубую точку заново.

Если вы захотите, чтобы ваше актуальное местоположение всегда отображалось вблизи от центра экрана, вам нужно будет постоянно отмечать эту точку при помощи анимации и получать регулярные обновления для явления, с которым вы работаете. В следующей версии этого упражнения мы повторно используем весь проект MyLocationDemo за исключением файла MyLocationDemoActivity.java. Новая версия MyLocationDemoActivity.java показана в листинге 7.20.

Листинг 7.20. Использование MyLocationOverlay и постоянное сохранение нашего местонахождения на виду

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;

import android.content.Context;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.widget.Toast;

public class MyLocationDemoActivity extends MapActivity {

    MapView mapView = null;
    MyLocationOverlay whereAmI = null;
    LocationManager locMgr = null;
    LocationListener locListener = null;

    @Override
    protected boolean isLocationDisplayed() {
```

```
        return whereAmI.isMyLocationEnabled();
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }

    /** Вызывается при первом создании явления. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView)findViewById(R.id.geoMap);
        mapView.setBuiltInZoomControls(true);
        mapView.getController().setZoom(15);

        whereAmI = new MyLocationOverlay(this, mapView);
        mapView.getOverlays().add(whereAmI);
        mapView.postInvalidate();

        locMgr = (LocationManager)getSystemService
                        (Context.LOCATION_SERVICE);
        locListener = new LocationListener()
        {
            public void onLocationChanged(Location location)
            {
                showLocation(location);
            }

            public void onProviderDisabled(String provider)
            {
            }

            public void onProviderEnabled(String provider)
            {
            }

            public void onStatusChanged(String provider,
                int status, Bundle extras)
            {
            }
        };
    }

    @Override
    public void onResume()
    {
        super.onResume();
```

```
        Location lastLoc =
              locMgr.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        showLocation(lastLoc);
        locMgr.requestLocationUpdates(
           LocationManager.GPS_PROVIDER,
           0, // минимальное время в миллисекундах
           0, // минимальное расстояние в метрах
           locListener);
        whereAmI.enableMyLocation();
        whereAmI.runOnFirstFix(new Runnable() {
           public void run() {
              mapView.getController().setCenter(whereAmI.getMyLocation());
           }
        });
    }

    @Override
    public void onPause()
    {
        super.onPause();
        locMgr.removeUpdates(locListener);
        whereAmI.disableMyLocation();
    }

    private void showLocation(Location location) {
        if (location != null)
        {
           double lat = location.getLatitude();
           double lng = location.getLongitude();
           GeoPoint myLocation = new GeoPoint(
              (int)(lat*1000000),
              (int)(lng*1000000));
           Toast.makeText(getBaseContext(),
              "New location latitude [" +
              lat + "] longitude [" + lng +"]",
              Toast.LENGTH_SHORT).show();
           mapView.getController().animateTo(myLocation);
        }
    }
}
```

На этот раз мы добавили наш собственный LocationListener, и теперь, по мере того как MyLocationOverlay получает обновления от поставщиков местоположения, мы также получаем обновления от GPS_PROVIDER. Поскольку мы будем получать одинаковые обновления, работа будет синхронизирована. Наш обратный вызов обращается к методу showLocation(), который перемещает карту, и наше текущее местоположение всегда остается на виду.

Применяя метод onResume(), мы тратим больше энергии, чем нужно; дополнительные вызовы показаны здесь только в качестве примера. Например, метод getLastKnownLocation() может вернуть null, если последнее местоположение неиз-

вестно, либо объект Location. Мы возвращаемся и вызываем метод showLocation() с тем же значением. Если у нас будут правильные данные о местоположении — хорошо, но в противном случае не произойдет ничего страшного. Мы также вызываем в рамках этого метода runOnFirstFix(), который выполняет практически ту же задачу. Если нам известно местоположение, карта немедленно переходит к нему. Разница заключается в следующем. Если мы не знаем нашего местоположения, то эта часть программы устанавливает Runnable так, чтобы, как только наше местоположение станет известно, оно оказалось в центре карты. Теперь запустим эту программу в эмуляторе, а затем будем посылать ей данные об обновлении местоположения через **Emulator Control**. Обратите внимание: в этом примере также используется Toast, он показывает точки, между которыми мы движемся. Наконец, еще раз отметим, что нулевые значения, присвоенные в нашем примере minTime и minDistance, в реальных программах не используются. Мы не будем получать обновленные данные при любом изменении местоположения, так как из-за этого батарея устройства очень быстро разрядится.

Резюме

В этой главе были рассмотрены две важные составляющие инструментария для разработки в Android: элементы для обеспечения безопасности приложений и службы, работающие в зависимости от местоположения устройства.

Вы узнали, что все программы в Android должны иметь цифровую подпись. Мы обсудили вопросы безопасности на этапе разработки программы в эмуляторе и Eclipse, а также подписывание программы Android перед релизом. Кроме того, мы поговорили о безопасности во время исполнения. Вы узнали, что инсталлятор Android запрашивает права доступа, требуемые вашей программе во время установки. Кроме того, мы показали, как определять права доступа, необходимые программе для работы, и как подписывать файл APK для развертывания программы.

Мы подробно рассмотрели, как работать с элементом управления MapView и с классом MapActivity. Мы начали изучение работы с картой с самых основ, а затем показали, как при помощи наложений размещать на карте маркеры. Кроме того, мы рассказали о геокодировании и о том, как этот процесс реализуется в фоновых потоках. Был рассмотрен класс LocationManager, предоставляющий подробную информацию о местонахождении при помощи поставщиков местоположения. Наконец, вы изучили, как показать актуальное расположение устройства на карте.

В следующей главе мы поговорим о создании служб в Android и об их использовании.

8 Создание и использование служб

Платформа Android располагает полномасштабным программным стеком. Это означает, что при работе с Android в вашем распоряжении будут операционная система, промежуточное программное обеспечение (middleware), а также рабочие приложения (к их числу относится, например, программа-номеронабиратель). Вдобавок к этому у вас есть инструментарий для разработки программ для данной платформы. Вы уже знаете, что в Android можно писать программы для непосредственного взаимодействия с пользователем через пользовательский интерфейс. Но мы еще не обсудили служебные программы, работающие в фоновом режиме, а также возможности создания компонентов, функционирующих в фоновом режиме.

В этой главе мы сосредоточимся на создании служб в Android и их использовании. Сначала мы поговорим о применении служб HTTP, потом перейдем к обсуждению межпроцессных взаимодействий (interprocess communication) — обмена информацией между программами, работающими на одном и том же устройстве.

Использование HTTP-служб

Приложения Android, как и любые другие программы для мобильных устройств, обычно невелики по размеру, но очень функциональны. Один из способов, которым достигается такая разнообразная функциональность на небольшом устройстве — получение программами информации из различных источников. Например, на T-Mobile G1 при поставке содержится программа Maps, оснащенная довольно совершенными картографическими функциями. Однако нам известно, что эта программа интегрирована с Google Maps и другими службами, благодаря которым и достигается такое совершенство.

Создаваемые вами программы, скорее всего, также будут активно использовать информацию, получаемую от других программ. Обычно для интеграции нескольких программ используется протокол HTTP. Например, в Интернете может быть доступен сервлет Java, предоставляющий службы, которые нужны вам для более эффективной работы одной из программ Android. Как применить этот сервлет в Android?

Интересно отметить, что в состав Android SDK входит клиент `HttpClient` для Apache (http://hc.apache.org/httpclient-3.x/), являющийся универсальным средством

для работы с J2EE. В Android SDK содержится версия HttpClient, модифицированная с учетом требований Android, но прикладные программные интерфейсы (API) очень похожи на соответствующие API из J2EE.

Клиент Apache HttpClient является комплексным и многосторонним. Хотя этот клиент полностью поддерживает протокол HTTP, вы, скорее всего, будете пользоваться методами HTTP GET и POST. В этом разделе мы обсудим, как делать вызовы GET и POST при помощи клиента HTTP.

Использование HttpClient для создания запросов HTTP GET

Ниже показана общая схема использования HttpClient.

1. Создание HttpClient (или получение имеющейся ссылки).
2. Инстанцирование нового HTTP-метода, например PostMethod или GetMethod.
3. Установка имен и значений параметров HTTP.
4. Выполнение вызова HTTP при помощи HttpClient.
5. Обработка отклика HTTP.

В листинге 8.1 показано, как выполнить запрос HTTP GET при помощи HttpClient.

ПРИМЕЧАНИЕ

Поскольку в коде заложена попытка выхода в Интернет, при использовании HTTP-вызовов с применением HttpClient необходимо добавить в файл описания право доступа android.permission.INTERNET.

Листинг 8.1. Использование HttpClient для получения запроса HTTP GET

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

public class TestHttpGet {

   public void executeHttpGet() throws Exception {
   BufferedReader in = null;
   try {
      HttpClient client = new DefaultHttpClient();
      HttpGet request = new HttpGet();
      request.setURI(new URI("http://code.google.com/android/"));
      HttpResponse response = client.execute(request);
      in = new BufferedReader
           (new InputStreamReader(response.getEntity()
```

```
                .getContent()));

        StringBuffer sb = new StringBuffer("");
        String line = "";
        String NL = System.getProperty("line.separator");
        while ((line = in.readLine()) != null) {
            sb.append(line + NL);
        }
        in.close();

        String page = sb.toString();
        System.out.println(page);
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

В HttpClient предусмотрены абстракции для описания HTTP-запросов различных типов, в частности HttpGet, HttpPost и т. д. В листинге 8.1 HttpClient используется для получения информации, содержащейся по ссылке **http://code.google.com/android/**. Для выполнения самого HTTP-запроса вызывается метод client.execute(). После выполнения запроса код считывает весь полученный ответ и помещает его в объект-строку (string). Обратите внимание — BufferedReader закрывается в блоке кода finally. Вместе с этим завершается и базовое HTTP-соединение.

Учтите, что класс из листинга 8.1 не является дополнением android.app.Activity. Иными словами, для работы с HttpClient вам не обязательно находиться в контексте определенного явления, так как HttpClient входит в пакет программ Android и этот клиент может использоваться и в контексте компонента Android (например, отдельно взятого явления), и как часть отдельного класса.

Код из листинга 8.1 выполняет HTTP-запрос, не передавая на сервер никаких параметров HTTP. Параметры «имя/значение» можно передавать вместе с запросом, добавляя пары «имя/значение» к URL. Пример показан в листинге 8.2.

Листинг 8.2. Добавление параметров к запросу HTTP GET

```
HttpGet method = new HttpGet("http://somehost/WS2/Upload.aspx?one=valueGoesHere");

client.execute(method);
```

При выполнении HTTP GET параметры (имена и значения) запроса передаются в URL. При передаче параметров таким способом вы сталкиваетесь с рядом огра-

ничений. Дело в том, что длина гиперссылки не должна превышать 2048 символов. Вместо HTTP GET можно использовать HTTP POST. Метод POST является более гибким — при его применении параметры передаются в теле запроса.

Использование HttpClient для создания запросов HTTP POST

Выполнение вызова HTTP POST очень напоминает вызов HTTP GET (листинг 8.3).

Листинг 8.3. Выполнение запроса HTTP POST при помощи HttpClient

```
import java.util.ArrayList;
import java.util.List;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

public class TestHttpPost
{
   public String executeHttpPost() throws Exception {
      BufferedReader in = null;
      try {
         HttpClient client = new DefaultHttpClient();
         HttpPost request = new HttpPost(
               "http://somewebsite/WS2/Upload.aspx");

         List<NameValuePair> postParameters =
               new ArrayList<NameValuePair>();
         postParameters.add(new BasicNameValuePair("one",
               "valueGoesHere"));
         UrlEncodedFormEntity formEntity = new UrlEncodedFormEntity(
               postParameters);

         request.setEntity(formEntity);
         HttpResponse response = client.execute(request);
         in = new BufferedReader(new
               InputStreamReader(response.getEntity().getContent()));

         StringBuffer sb = new StringBuffer("");
         String line = "";
         String NL = System.getProperty("line.separator");
         while ((line = in.readLine()) != null) {
            sb.append(line + NL);
         }
```

```
            in.close();

            String result = sb.toString();
            return result;
        } finally {
            if (in != null) {
                try {
                    in.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Для выполнения запроса HTTP POST с помощью HttpClient нужно вызвать метод execute() этого клиента по отношению к экземпляру HttpClient. При выполнении вызовов HTTP POST параметры форм вида «имя/значение» обычно кодируются как часть URL, передаваемой при запросе HTTP. Чтобы осуществить такую операцию при помощи HttpClient, нужно создать список экземпляров объектов NameValuePair, а затем заключить этот список в объект UrlEncodedFormEntity. Объект NameValuePair содержит в себе комбинацию «имя/значение», а класс UrlEncodedFormEntity умеет кодировать список объектов NameValuePair для последующего их применения в вызовах HTTP (обычно используются вызовы POST). После создания UrlEncodedFormEntity можно задать UrlEncodedFormEntity в качестве типа сущности (entity type) HttpPost, а затем выполнить запрос.

В листинге 8.3 создается HttpClient, а затем инстанцируется HttpPost с URL, указывающей на пункт назначения HTTP Endpoint (конечная точка HTTP). Далее создается список объектов NameValuePair, который заполняется параметрами отдельных пар «имя/значение». Именем параметра будет one, а значением — valueGoesHere. Затем создается экземпляр UrlEncodedFormEntity. Его конструктору передается список NameValuePairobjects. Наконец, мы вызываем метод setEntity() запроса POST и выполняем запрос при помощи экземпляра HttpClient.

Здесь описаны далеко не все возможности HTTP POST. Он позволяет передавать простые параметры «имя/значение» (такой случай описан в листинге 8.3), а также сложные параметры, например файлы. HTTP POST также может работать с запросами другого формата, который называется *multipart POST*. Используя такой тип POST, вы можете передавать с запросом не только пары «имя/значение», но и любые файлы. К сожалению, multipart POST напрямую не поддерживается в той версии HttpClient, которая входит в состав Android. Для выполнения вызовов multipart POST в Android нужно освоить еще три свободно распространяемых проекта Apache: Apache Commons IO, Mime4j и HttpMime. Эти проекты можно скачать на следующих сайтах:

- *Commons IO* — http://commons.apache.org/io/;
- *Mime4j* — http://james.apache.org/mime4j/;
- *HttpMime* — http://hc.apache.org/httpcomponentsclient/httpmime/index.html.

Можно также посетить следующий сайт, на котором есть файлы JAR, необходимые для выполнения `multipart POST` в Android: **http://www.apress.com/book/view/1430226595**.

В листинге 8.4 показано использование `multipart POST` в Android.

Листинг 8.4. Выполнение вызова multipart POST

```
import java.io.ByteArrayInputStream;
import java.io.InputStream;

import org.apache.commons.io.IOUtils;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.mime.MultipartEntity;
import org.apache.http.entity.mime.content.InputStreamBody;
import org.apache.http.entity.mime.content.StringBody;
import org.apache.http.impl.client.DefaultHttpClient;

import android.app.Activity;

public class TestMultipartPost extends Activity
{
   public void executeMultipartPost()throws Exception
   {
      try {
         InputStream is = this.getAssets().open("data.xml");
         HttpClient httpClient = new DefaultHttpClient();
         HttpPost postRequest =
               new HttpPost("http://192.178.10.131/WS2/Upload.aspx");

         byte[] data = IOUtils.toByteArray(is);

         InputStreamBody isb = new InputStreamBody(new
               ByteArrayInputStream(data),"uploadedFile");
         StringBody sb1 = new StringBody("someTextGoesHere");
         StringBody sb2 = new StringBody("someTextGoesHere too");

         MultipartEntity multipartContent = new MultipartEntity();
         multipartContent.addPart("uploadedFile", isb);
         multipartContent.addPart("one", sb1);
         multipartContent.addPart("two", sb2);

         postRequest.setEntity(multipartContent);
         HttpResponse res =httpClient.execute(postRequest);
         res.getEntity().getContent().close();
      } catch (Throwable e)
      {
         // здесь обрабатывается исключение
      }
   }
}
```

ПРИМЕЧАНИЕ

В примере с multipart использованы файлы JAR, которые не входят в состав рабочей среды Android. Чтобы гарантировать, что файлы JAR будут входить в единый пакет вместе с вашим файлом APK, в Eclipse их необходимо добавить как внешние JAR-файлы. Для этого щелкните правой кнопкой мыши в Eclipse на вашем проекте, выберите Properties (Свойства), затем Java Class Path (Путь к классу Java), вкладку Libraries (Библиотеки), а на ней — параметр Add External JARs (Добавить внешние JAR).

После выполнения этих этапов файлы JAR будут доступны и во время компиляции, и во время исполнения.

Для выполнения `multipart POST` необходимо создать `HttpPost` и вызвать его метод `setEntity()` с экземпляром `MultipartEntity` (а не с `UrlEncodedFormEntity`, который мы создавали для работы с параметрами вида «имя/значение»). `MultipartEntity` представляет собой тело запроса `multipart POST`. Как видите, создается экземпляр `MultipartEntity`, а затем для каждой части вызывается метод `addPart()`. В листинге 8.4 к запросу добавляется три части: две строковые и один файл XML.

Наконец, если вы пишете программу, при работе с которой необходимо посылать `multipart POST` на веб-ресурс, вам, вероятно, потребуется усовершенствовать показанное здесь решение, внедрив формальную реализацию (dummy implementation) службы на вашей локальной рабочей станции. Работая с программами на такой рабочей станции, вы можете получить доступ к локальной машине по `localhost` или по IP-адресу 127.0.0.1. Но при работе с программами для Android `localhost` или 127.0.0.1 не используется, так как у эмулятора будет собственный `localhost`. Чтобы сделать ссылку на вашу рабочую станцию из программы, работающей в эмуляторе, нужно использовать IP рабочей станции (если вы не помните, что такое IP рабочей станции, обратитесь к главе 2, где этот вопрос подробно объяснен). Измените листинг 8.4, подставив в качестве IP адрес вашей рабочей станции.

Но как быть с SOAP? Многие веб-службы в Интернете работают на базе SOAP, но пока Google не предоставляет в Android возможность для непосредственного вызова SOAP-служб. Вместо этого Google предпочитает работать с REST-подобными службами, видимо, для того, чтобы уменьшить объем вычислений, которые необходимо выполнять на клиентском устройстве. Однако при подобном уменьшении объема вычислений разработчик вынужден выполнять больше работы при отправке данных и при синтаксическом разборе полученного результата. В идеальном случае у вас будет несколько возможностей взаимодействия с веб-службами. Некоторые специалисты при написании SOAP-клиентов для Android используют инструментарий разработчика kSOAP2. Этот инструментарий мы не будем рассматривать; если он вам интересен, посетите сайт **http://ksoap2.sourceforge.net/**.

Работа с исключениями

Работа с исключениями происходит в любой программе, но в ПО, использующем внешние службы (в частности, HTTP-службы), исключения требуют особого внимания, поскольку возможность возникновения ошибок очень высока. Существует несколько типов исключений, с которыми вы можете столкнуться при работе с HTTP-службами. Это исключения передачи (transport exceptions), исключения

протоколов (protocol exceptions) и задержки (timeouts). Необходимо понимать, в каких случаях могут возникать такие исключения.

Исключения передачи могут появиться по нескольким причинам, наиболее вероятная из которых — плохое соединение мобильного устройства с сетью. Исключения протоколов происходят на уровне протокола HTTP. К ним относятся ошибки аутентификации, неверные cookies и т. д. Вы вполне можете встретиться с исключениями протоколов в тех случаях, когда, например, в HTTP-запросе требуется передать учетные данные для входа в систему, а сделать этого не удается. Задержки при выполнении HTTP-вызовов делятся на две категории: задержки соединения (connection timeouts) и задержки ожидания данных (socket timeout). Задержка соединения может возникнуть, если `HttpClient` не может связаться с HTTP-сервером — например, если ссылка неверна или HTTP-сервер недоступен. Задержки ожидания данных могут появляться, если `HttpClient` не может дать ответ в течение заданного периода. Иными словами, `HttpClient` смог связаться с сервером, но сервер не может дать ответ в течение заданного временного промежутка.

Теперь когда мы вкратце описали типы исключений, которые могут возникнуть, разберемся — что с ними делать? К счастью, `HttpClient` достаточно надежен и выполняет большую часть работы за вас. Любые исключения, которые могут вам помешать, устраняются без проблем. `HttpClient` сам решает проблемы с исключениями передачи, идентифицируя проблемы, возникающие на пути следования данных, и посылая повторные запросы (повторные запросы — очень действенное средство борьбы с исключениями передачи). Почти все исключения протоколов можно устранить уже на этапе разработки программы. Однако с задержками вы будете сталкиваться не так уж и редко. Эффективный способ борьбы с задержками обоих упомянутых типов (соединения и ожидания данных) — заключать метод `execute()` HTTP-запроса в `try/catch`, а при возникновении ошибки повторять запрос. Этот прием показан в листинге 8.5.

Листинг 8.5. Реализация простого механизма отправки повторных запросов при работе с задержками

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URI;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;

public class TestHttpGet {

   public String executeHttpGetWithRetry() throws Exception {
      int retry = 3;

      int count = 0;
      while (count < retry) {
         count += 1;
```

```
        try {
            String response = executeHttpGet();
            /**
            * Если мы дошли до этого шага,
            * это означает, что работа успешно завершена.
            */
            return response;
        } catch (Exception e) {
            /**
            * Если мы исчерпали лимит повторных попыток.
            */
            if (count < retry) {
            /**
            * У нас еще есть попытки, поэтому заносим сообщение
            * в журнал и пробуем снова.
            */
            System.out.println(e.getMessage());
            } else {
                System.out.println("could not succeed with retry...");
                throw e;
            }
        }
    }
    return null;
}

public String executeHttpGet() throws Exception {
    BufferedReader in = null;
    try {
        HttpClient client = new DefaultHttpClient();
        HttpGet request = new HttpGet();
        request.setURI(new URI("http://code.google.com/android/"));
        HttpResponse response = client.execute(request);
        in = new BufferedReader(new
              InputStreamReader(response.getEntity().getContent()));

        StringBuffer sb = new StringBuffer("");
        String line = "";
        String NL = System.getProperty("line.separator");
        while ((line = in.readLine()) != null) {
            sb.append(line + NL);
        }
        in.close();

        String result = sb.toString();
        return result;
    } finally {
        if (in != null) {
            try {
                in.close();
            } catch (IOException e) {
```

```
                e.printStackTrace();
            }
        }
    }
  }
}
```

В коде листинга 8.5 показано, как внедрить в программу простой механизм повторных запросов, чтобы справляться с задержками при HTTP-вызовах. В листинге приведены два метода: один выполняет HTTP GET (executeHttpGet()), а другой служит оболочкой первого метода и содержит логику повторных попыток (executeHttpGetWithRetry()). Такая логика (retry logic) очень проста. Мы устанавливаем для количества попыток выполнения запроса значение 3, а затем запускаем цикл while. В этом цикле выполняется запрос. Обратите внимание: запрос заключен в блок try/catch, и в части catch система проверяет, не исчерпали ли мы отпущенный лимит повторных попыток.

При использовании HttpClient в реальных приложениях вы также должны уметь решать задачи, связанные с многопоточностью (multithreading issues). Рассмотрим их.

Решение задач, связанных с многопоточностью

В примерах, показанных выше, для каждого запроса создается новый HttpClient. Но на практике нужно создавать один HttpClient для всей программы и использовать его для обеспечения всего обмена информацией по HTTP. Имея один HttpClient, обслуживающий все HTTP-запросы, обращайте внимание на проблемы, связанные с многопоточностью, которые могут возникнуть, если через один и тот же HttpClient одновременно отправляется несколько запросов. Правда, в HttpClient есть функция, облегчающая вам работу. Достаточно создать один DefaultHttpClient, используя ThreadSafeClientConnManager, как это показано в листинге 8.6.

Листинг 8.6. Создание HttpClient для работы с многопоточностью

```
// ApplicationEx.java
import org.apache.http.HttpVersion;
import org.apache.http.client.HttpClient;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.impl.conn.tsccm.ThreadSafeClientConnManager;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpParams;
import org.apache.http.params.HttpProtocolParams;
import org.apache.http.protocol.HTTP;

import android.app.Application;
```

```
import android.util.Log;

public class ApplicationEx extends Application
{
   private static final String TAG = "ApplicationEx";
   private HttpClient httpClient;

   @Override
   public void onCreate()
   {
      super.onCreate();

      httpClient = createHttpClient();

      }

   @Override
   public void onLowMemory()
   {
      super.onLowMemory();
      shutdownHttpClient();
   }

   @Override
   public void onTerminate()
   {
      super.onTerminate();
      shutdownHttpClient();
   }

   private HttpClient createHttpClient()
   {
      Log.d(TAG,"createHttpClient()...");
      HttpParams params = new BasicHttpParams();
      HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
      HttpProtocolParams.setContentCharset(params,
               HTTP.DEFAULT_CONTENT_CHARSET);
      HttpProtocolParams.setUseExpectContinue(params, true);

      SchemeRegistry schReg = new SchemeRegistry();
      schReg.register(new Scheme("http",
               PlainSocketFaçtory.getSocketFactory(), 80));
      schReg.register(new Scheme(„https",
      SSLSocketFactory.getSocketFactory(), 443));
      ClientConnectionManager conMgr = new
               ThreadSafeClientConnManager(params,schReg);

      return new DefaultHttpClient(conMgr, params);
   }

   public HttpClient getHttpClient() {
      return httpClient;
```

```
    }
    private void shutdownHttpClient()
    {
        if(httpClient!=null && httpClient.getConnectionManager()!=null)
        {
            httpClient.getConnectionManager().shutdown();
        }
    }
}

// HttpActivity.java

import java.net.URI;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.util.EntityUtils;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class HttpActivity extends Activity
{
    /** Вызывается при первом создании явления. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        Log.d("ServicesDemoActivity", "a debug statement");
        getHttpContent();
    }
    public void getHttpContent()
    {
        try {
            ApplicationEx app = (ApplicationEx)this.getApplication();
            HttpClient client = app.getHttpClient();
            HttpGet request = new HttpGet();
            request.setURI(new URI("http://www.google.com/"));
            HttpResponse response = client.execute(request);

            String page=EntityUtils.toString(response.getEntity());
            System.out.println(page);
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

Обратите внимание — при переопределении (override) или дополнении (extend) заданного по умолчанию объекта приложения, нужно также модифицировать узел этой программы, расположенный в файле AndroidManifest.xml, устанавливая для этого атрибут android:name следующим образом:

```
<application android:icon="@drawable/icon"
android:label="@string/app_name"
android:name="ApplicationEx">
```

Если в программе потребуется сделать не пару вызовов HTTP, а несколько больше, нужно создать такой HttpClient, который будет обслуживать все HTTP-запросы. Один из способов сделать это — воспользоваться тем фактом, что с каждым приложением Android по умолчанию связан соответствующий объект-приложение. По умолчанию, если вы не определите объект-приложение специально, Android использует android.app.Application. С объектом «приложение» связан один интересный момент: каждому вашему приложению всегда соответствует только один объект и все компоненты имеют к нему доступ (через глобальный контекстный объект).

Например, можно вызвать из класса явления метод getApplication(), чтобы получить для конкретной программы объект-приложение. Идея заключается в том, что поскольку программа является синглтоном и всегда доступна, мы можем дополнить его класс и создать в нем HttpClient. Затем мы предоставим для всех компонентов метод доступа (accessor method), чтобы эти компоненты могли обращаться к HttpClient. Именно это мы и сделали в листинге 8.6. В нем мы определили два класса (каждый из них должен находиться в отдельном Java-файле). Один из файлов — наш специальный объект приложения, а второй — типичный объект: класс activity. В классе ApplicationEx мы дополняем android.app.Application, а затем создаем HttpClient в методе onCreate(). Затем класс предоставляет метод доступа для компонентов, чтобы они могли получать ссылки на клиент. В классе HttpActivity мы получаем ссылку на глобальный объект приложения и помещаем ее в класс ApplicationEx. Затем вызываем метод getHttpClient() и с его помощью делаем HTTP-вызов.

Теперь рассмотрим метод createHttpClient() класса ApplicationEx. Он отвечает за создание синглтона HttpClient. Обратите внимание: при инстанцировании DefaultHttpClient() мы передаем ClientConnectionManager, который отвечает за управление HTTP-соединениями при работе с HttpClient. Поскольку мы хотим использовать один HttpClient для всех HTTP-запросов, мы создаем ThreadSafeClientConnManager.

ПРИМЕЧАНИЕ

Закончив работу с диспетчером соединений, следует применить к нему метод shutdown(), как это показано в листинге 8.6.

На этом наш разговор об использовании HTTP-служб с HttpClient завершается. В следующих разделах мы сосредоточимся на другом, не менее интересном компоненте платформы Android: фоновых службах и службах длительного выполнения. Хотя это и не очевидно, но процессы создания HTTP-запросов и написания служб

Android в значительной мере взаимосвязаны. Эта связь выражается в большом объеме работ по интеграции внутрисистемных служб Android. Возьмем, к примеру, простой почтовый клиент. В устройстве Android программа такого типа, скорее всего, будет состоять из двух частей: пользовательского интерфейса и компонента для поллинга с целью получения электронных сообщений. Вероятно, поллинг будет выполняться фоновой службой. Компонент, проверяющий наличие новых электронных сообщений, будет службой Android, которая, в свою очередь, при работе будет использовать HttpClient.

Теперь перейдем к написанию служб.

Обеспечение межпроцессного обмена информацией

В Android поддерживается работа со службами. *Службы* — это компоненты, работающие в фоновом режиме, без пользовательского интерфейса. Подобные компоненты имеются и в Windows, и в Unix. Подобно службам из этих операционных систем, службы Android всегда доступны, но могут в течение определенных периодов оставаться без дела.

В Android используются службы двух типов: *локальные* (local) и *удаленные* (remote). Локальная отличается тем, что к ней закрыт доступ для других программ, работающих в устройстве. Обычно такие службы просто поддерживают работу программы, которая является для них несущей (hosting). Удаленная служба доступна не только для несущей программы, но и для всех остальных программ, работающих в устройстве. Удаленные службы определяют себя для клиентов, используя язык описания интерфейса Android (AIDL).

Для начала напишем простую службу.

Создание простой службы

Чтобы построить службу, необходимо дополнить класс android.app.Service и поместить запись о конфигурации службы в файл описания приложения. Соответствующий пример показан в листинге 8.7.

Листинг 8.7. Определение простой службы в Android

```
import android.app.Service;
public class TestService1 extends Service
{
   private static final String TAG = "TestService1";

   @Override
   public void onCreate() {
      Log.d(TAG, "onCreate");
      super.onCreate();
   }

   @Override
```

```
   public IBinder onBind(Intent intent) {
      Log.d(TAG, "onBind");
      return null;
   }
}

// Запись с определением службы: в файле AndroidManifest.xml должна
// представлять собой дочерний элемент относительно <application>.
<service android:name="TestService1"></service>
```

Служба из листинга 8.7 не предназначена для реального использования, но с ее помощью можно понять, как происходит определение службы. Для создания службы пишется класс, дополняющий android.app.Service и реализующий метод onBind(). Затем запись с описанием службы помещается в файл AndroidManifest.xml. Так определяется служба. Следующий очевидный вопрос заключается в том, как вызвать службу. Ответ зависит от того, с каким именно клиентом работает служба, и требует более глубокого изучения служб.

Службы в Android

Чтобы лучше понять природу служб, рассмотрим общие методы android.app.Service (листинг 8.8).

Листинг 8.8. Общие методы службы

```
Application getApplication();
abstract IBinder onBind(Intent intent);
void  onConfigurationChanged(Configuration newConfig);
void  onCreate();
void  onDestroy();
void  onLowMemory();
void  onRebind(Intent intent);
void  onStart(Intent intent, int startId);
boolean onUnbind(Intent intent);
final void     setForeground(boolean isForeground);
final void     stopSelf();
final void     stopSelf(int startId);
final boolean  stopSelfResult(int startId);
```

Метод getApplication() возвращает приложение, в котором используется конкретная служба. Метод onBind() предоставляет интерфейс для работы с внешними приложениями, которые работают в том же устройстве. Через этот интерфейс происходит обмен информацией между приложениями и данной службой. onConfigurationChanged() позволяет службе изменить свою конфигурацию в случае, если изменится конфигурация устройства.

Система вызывает onCreate() при первом создании службы, но до вызова onStart(). Этот процесс, напоминающий создание явления, обеспечивает для службы возможность одноразовой инициализации при запуске (см. раздел «Жизненный цикл приложения» в главе 2, там детально описан процесс создания явления). Например, создавать фоновый поток следует методом onCreate(), а оста-

навливать работу потока нужно при помощи onDestroy(). Система вызывает onCreate(), затем onStart(), а при завершении работы службы — onDestroy(). В методе onDestroy() предусмотрен механизм окончательного сглаживания (final cleanup) до завершения работы службы.

Обратите внимание на то, что onStart(), onCreate() и onDestroy() вызываются самой системой; вы сами не должны их вызывать. Более того, при переопределении любого из методов on*() в классе вашей службы обязательно сами вызывайте версию суперкласса. Различные версии stopSelf() позволяют программе остановить работу службы. Об этих и других методах мы поговорим в подразделе «Локальные службы» далее.

В Android службы поддерживаются по двум основным причинам. Во-первых, они упрощают выполнение задач в фоновом режиме; во-вторых, обеспечивают межпроцессный обмен информацией между приложениями, работающими в одном и том же устройстве. Эти причины соотносятся с двумя типами служб, поддерживаемых в Android: локальными и удаленными. Примером из первой группы может быть локальная служба, представляющая собой часть почтового приложения, — об этом мы говорили выше. Служба будет посылать запросы на почтовый сервер относительно того, не пришли ли новые сообщения, и уведомлять пользователя о получении новых писем. Примером из второй группы может быть маршрутизатор. Предположим, в одном и том же устройстве у нас работает несколько программ, а нам нужна служба, которая будет принимать сообщения и направлять их в различные места назначения. Чтобы не повторять эту логику в каждой отдельной программе, можно написать удаленный маршрутизатор и организовать обмен информацией между ним и программами.

Существуют важные различия между локальными и удаленными службами. В частности, если служба используется только компонентами одного и того же процесса (для выполнения определенных задач в фоновом режиме), то клиенты должны запускать службу при помощи метода Context.startService(). Служба такого типа является локальной, так как ее цель — выполнять в фоновом режиме задачи, необходимые для работы несущего приложения. Если служба поддерживает метод onBind(), то она является удаленной, и может быть вызвана путем межпроцессного обмена информацией (Context.bindService()). Удаленные службы также именуются ***службами с поддержкой AIDL***, так как клиенты связываются с ними с применением языка AIDL.

Хотя интерфейс android.app.Service и поддерживает как локальные, так и удаленные службы, не стоит создавать один экземпляр службы для одновременной поддержки обоих типов. Причина в том, что у каждого типа службы есть заранее заданный жизненный цикл; и хотя смешивание двух типов допускается, это чревато ошибками.

Перейдем к подробному изучению обоих типов служб. Начнем с беседы о локальных службах, а потом перейдем к обсуждению удаленных (поддерживающих язык AIDL). Как было указано выше, локальными являются службы, которые вызываются только приложением, несущим их. Удаленные службы поддерживают механизм удаленного вызова процедур (RPC). Эти службы позволяют внешним клиентам, работающим на одном и том же устройстве, связываться со службой и пользоваться ее функциями.

ПРИМЕЧАНИЕ

Службы второго типа известны в Android под несколькими названиями: удаленные службы, службы с поддержкой AIDL, внешние службы и службы для удаленного вызова процедур. Все эти термины относятся к одному и тому же типу служб — доступ к которым открыт для всех приложений, работающих в устройстве.

Локальные службы

Локальные службы запускаются методом `Context.startService()`. После запуска службы этого типа продолжают работу, пока клиент не вызовет `Context.stopService()` либо сама служба не вызовет `stopSelf()`. Обратите внимание: при вызове `Context.startService()` система инстанцирует службу и вызывает для нее метод `onStart()`. Помните, что при вызове `Context.startService()` после запуска службы (то есть когда служба уже работает), не будет создан новый экземпляр службы, а последует вызов метода `onStart()` для этой службы. Ниже приведено два примера локальных служб:

- служба для нахождения данных в сети (например, в Интернете), работающая на базе таймера (применяется для закачки или скачивания информации);
- служба для выполнения задач, позволяющая явлениям вашей программы отправлять задачи и располагать их в очередь для последующей обработки.

В листинге 8.9 показана служба, предназначенная для выполнения фоновых задач. Здесь содержатся все компоненты, необходимые для создания и использования службы: `BackgroundService.java`, сама служба; `MainActivity.java`, класс явления для вызова службы, а также файл `main.xml`, шаблон для явления.

Листинг 8.9. Внедрение локальной службы

```
// BackgroundService.java

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class BackgroundService extends Service
{
   private NotificationManager notificationMgr;

   @Override
   public void onCreate() {
      super.onCreate();

      notificationMgr =(NotificationManager)getSystemService(
                  NOTIFICATION_SERVICE);

      displayNotificationMessage("starting Background Service");

      Thread thr = new Thread(null, new ServiceWorker(),
```

```
                        "BackgroundService");
      thr.start();

   }

   class ServiceWorker implements Runnable
   {
      public void run() {
         // здесь выполняется фоновая обработка

         // завершение работы службы после выполнения задачи
         // BackgroundService.this.stopSelf();
      }
   }

   @Override
   public void onDestroy()
   {
      displayNotificationMessage("stopping Background Service");
      super.onDestroy();

   }

   @Override
   public void onStart(Intent intent, int startId) {
      super.onStart(intent, startId);

   }

   @Override
   public IBinder onBind(Intent intent) {
      return null;
   }

   private void displayNotificationMessage(String message)
   {

      Notification notification = new Notification(R.drawable.note,
               message,System.currentTimeMillis());
      PendingIntent contentIntent = PendingIntent.getActivity
               (this, 0, new Intent(this, MainActivity.class), 0);
      notification.setLatestEventInfo(this, "Background Service",
               message,contentIntent);
      notificationMgr.notify(R.id.app_notification_id, notification);
   }
}

// MainActivity.java

import android.app.Activity;
import android.content.Intent;
```

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity
{
   private static final String TAG = "MainActivity";

   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      Log.d(TAG, "starting service");

      Button bindBtn = (Button)findViewById(R.id.bindBtn);
      bindBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View arg0) {
            startService(new Intent(MainActivity.this,
                  BackgroundService.class));
         }});

      Button unbindBtn = (Button)findViewById(R.id.unbindBtn);
      unbindBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View arg0) {
            stopService(new Intent(MainActivity.this,
                     BackgroundService.class));
         }});
   }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button android:id="@+id/bindBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
```

```
    android:text="Bind"
    />

    <Button android:id="@+id/unbindBtn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="UnBind"
    />
</LinearLayout>
```

Для запуска этого примера необходимо создать службу BackgroundService.java, класс явления MainActivity.java и файл шаблона main.xml. Кроме того, вам потребуется создать пиктограмму note и поместить ее в каталог drawable вашего проекта. Для диспетчера уведомлений понадобится уникальный ID на уровне приложения (целое число). Чтобы создать уникальный ID, достаточно добавить элемент ID в файл строковых ресурсов, находящийся в /res/values/strings.xml. Уникальный ID передается диспетчеру уведомлений при вызове метода notify(). В нашем примере используется следующий вариант:

```
<item type="id" name="app_notification_id"/>
```

Наконец, нужно добавить тег <service android:name="BackgroundService"/> в файл AndroidManifest.xml. Этот тег будет дочерним относительно <application>.

Обратите внимание: в листинге 8.9 используется специальное явление, предназначенное для взаимодействия со службой, но службу может использовать и любой другой компонент приложения. К таким компонентам относятся другие службы, явления, родовые классы (generic classes) и т. д. В примере создается пользовательский интерфейс с двумя кнопками, которые называются **Bind** (Связать) и **UnBind** (Разъединить). При нажатии **Bind** (Связать) служба запускается (происходит вызов метода startService()); при нажатии **UnBind** (Разъединить) служба останавливается (происходит вызов stopService()). Теперь разберем основную часть этого примера: BackgroundService.

BackgroundService — это типичная служба, используемая компонентами программы, несущей эту службу. Другими словами, именно то приложение, которое выполняет данную службу, ее и использует. Поскольку служба не поддерживает клиентов, которые являются внешними относительно данного процесса, это локальна служба. И так как эта служба именно локальная, а не удаленная, она возвращает в методе bind() значение null. Следовательно, единственный способ выполнить привязку к данной службе — вызвать Context.startService(). К числу важнейших методов локальной службы относятся onCreate(), onStart(), stop*() и onDestroy().

В методе onCreate() службы BackgroundService мы создаем поток, в котором выполняется основная часть работы службы. Основной поток приложения нужен нам для работы с пользовательским интерфейсом, поэтому для выполнения служебных задач создается второй поток. Отметим также, что для создания и запуска потока применяется onCreate(), а не onStart(). Мы поступаем так, поскольку onCreate() вызывается однократно, а нам нужно, чтобы поток создавался только однажды на протяжении всего жизненного цикла приложения. OnStart() можно вызывать многократно, поэтому в данном случае он нам не подходит. Если мы внедрим метод

потока `run()`, то также не сделаем ничего полезного, но именно отсюда будут происходить вызовы HTTP, запросы к базе данных и т. д.

`BackgroundService` также использует класс `NotificationManager` — для отправки пользователю уведомлений о запуске и остановке работы службы. Это единственный способ, которым локальная служба может передавать информацию пользователю. Для отправки уведомлений пользователю нужно получить диспетчер уведомлений. Это делается путем вызова `getSystemService(NOTIFICATION_SERVICE)`. Сообщения диспетчера уведомлений появляются в строке состояния.

На этом мы завершаем изучение локальных служб. Займемся службами более сложного типа, которые работают с AIDL.

Службы AIDL

В предыдущем подразделе было показано, как написать службу Android, которая будет применяться тем приложением, в котором она была создана. Далее будет показано, как написать службу, которая может использоваться другими процессами посредством вызова удаленных процедур (Remote Procedure Calls). Как и во многих других решениях, работающих на основе RPC, в Android применяется язык описания интерфейса (IDL), в котором определяется интерфейс для работы с клиентами. В Android язык описания интерфейса называется Android Interface Definition Language (AIDL). Чтобы создать удаленную службу, выполните следующие шаги.

1. Создайте файл AIDL, в котором ваш интерфейс будет описываться для клиентов. В файле AIDL используется синтаксис Java, файл имеет расширение AIDL. Используйте то же имя пакета, что и в проекте Android.
2. Добавьте файл AIDL в проект Eclipse, в каталог `src`. Плагин Android Eclipse вызовет компилятор AIDL для создания интерфейса Java для файла AIDL (компилятор AIDL вызывается в процессе сборки).
3. Внедрите службу и возвратите интерфейс при помощи метода `onBind()`.
4. Добавьте конфигурацию службы в файл `AndroidManifest.xml`.

В следующих подразделах будет подробно рассмотрено выполнение каждого шага.

Описание служебного интерфейса на AIDL

Рассмотрим работу с удаленной службой на примере программы-котировщика. Такая служебная программа предоставляет метод, который берет биржевой код и возвращает значение котировки. Чтобы создать удаленную службу в Android, сначала нужно описать интерфейс службы на языке AIDL. В листинге 8.10 показано такое описание `IStockQuoteService`.

Листинг 8.10. Описание служебной программы-котировщика на языке AIDL

```
// это файл IStockQuoteService.aidl
package com.androidbook.stockquoteservice;
interface IStockQuoteService
{
```

```
  double getQuote(String ticker);
}
```

IStockQuoteService принимает биржевой символ в виде строки и возвращает текущее значение котировки как число с плавающей запятой. При создании файла AIDL плагин Android Eclipse запускает компилятор AIDL для обработки этого файла (операция происходит в процессе сборки). Если компиляция AIDL-файла пройдет удачно, компилятор сгенерирует интерфейс Java, подходящий для вызова удаленных процедур. Обратите внимание: сгенерированный файл будет находиться в пакете, названном по имени вашего AIDL-файла; в данном случае — com.androidbook.stockquoteservice.

В листинге 8.11 показан сгенерированный файл Java для интерфейса IStockQuoteService. Сгенерированный файл будет помещен в основной каталог проекта Eclipse.

Листинг 8.11. Сгенерированный компилятором файл Java

```
/*
* Этот файл создан автоматически. НЕ ИЗМЕНЯТЬ.
* Оригинал файла: C:\\android\\StockQuoteService\\src\\com\\androidbook\\➥
stockquoteservice\\IStockQuoteService.aidl
*/
package com.androidbook.stockquoteservice;
import java.lang.String;
import android.os.RemoteException;
import android.os.IBinder;
import android.os.IInterface;
import android.os.Binder;
import android.os.Parcel;
public interface IStockQuoteService extends android.os.IInterface
{
/** Локальный класс-заглушка для реализации IPC. */
public static abstract class Stub extends android.os.Binder implements ➥
com.androidbook.stockquoteservice.IStockQuoteService
{
private static final java.lang.String DESCRIPTOR = ➥
"com.androidbook.stockquoteservice.IStockQuoteService";
/** Создание заглушки и прикрепление ее к интерфейсу. */
public Stub()
{
this.attachInterface(this, DESCRIPTOR);
}
/**
* Помещение объекта IBinder в интерфейс IStockQuoteService,
* при необходимости — генерирование прокси.
*/
public static com.androidbook.stockquoteservice.IStockQuoteService ➥
asInterface(android.os.IBinder obj)
{
if ((obj==null)) {
return null;
}
```

```
android.os.IInterface iin = (android.os.IInterface)obj.queryLocalInterface(DESCRIP
TOR);
if (((iin!=null)&&(iin instanceof
com.androidbook.stockquoteservice.IStockQuoteService))) {
return ((com.androidbook.stockquoteservice.IStockQuoteService)iin);
}
return ((com.androidbook.stockquoteservice.IStockQuoteService)iin);
}
return new com.androidbook.stockquoteservice.IStockQuoteService.Stub.Proxy(obj);
}
public android.os.IBinder asBinder()
{
return this;
}
@Override public boolean onTransact(int code, android.os.Parcel data,➡
   android.os.Parcel reply, int flags) throws android.os.RemoteException
{
switch (code)
{
case INTERFACE_TRANSACTION:
{
reply.writeString(DESCRIPTOR);
return true;
}
case TRANSACTION_getQuote:
{
data.enforceInterface(DESCRIPTOR);
java.lang.String _arg0;
_arg0 = data.readString();
double _result = this.getQuote(_arg0);
reply.writeNoException();
reply.writeDouble(_result);
return true;
}
}
return super.onTransact(code, data, reply, flags);
}
private static class Proxy implements
com.androidbook.stockquoteservice.IStockQuoteService
{
private android.os.IBinder mRemote;
Proxy(android.os.IBinder remote)
{
mRemote = remote;
}
public android.os.IBinder asBinder()
{
return mRemote;
}
public java.lang.String getInterfaceDescriptor()
{
```

```
return DESCRIPTOR;
}
public double getQuote(java.lang.String ticker) throws android.os.RemoteException
{
android.os.Parcel _data = android.os.Parcel.obtain();
android.os.Parcel _reply = android.os.Parcel.obtain();
double _result;
try {
_data.writeInterfaceToken(DESCRIPTOR);
_data.writeString(ticker);
mRemote.transact(Stub.TRANSACTION_getQuote, _data, _reply, 0);
_reply.readException();
_result = _reply.readDouble();
}
finally {
_reply.recycle();
_data.recycle();
}
return _result;
}
}
static final int TRANSACTION_getQuote = (IBinder.FIRST_CALL_TRANSACTION + 0);
}
public double getQuote(java.lang.String ticker) throws android.os.RemoteException;
}
```

Обратите внимание на следующие важные аспекты, касающиеся сгенерированных классов.

- Интерфейс, который мы описали в файле AIDL, интегрируется в виде интерфейса в сгенерированный код (речь идет об интерфейсе IStockQuoteService).
- Абстрактный класс static final под названием Stub (заглушка) дополняет android.os.Binder и реализует IStockQuoteService. Особо отметим, что этот класс является абстрактным.
- Внутренний класс под названием Proxy реализует IStockQuoteService, который проксирует класс Stub.
- Файл AIDL должен находиться в пакете, в котором система ожидает найти сгенерированные файлы (в соответствии с описанием пакета, данным в файле AIDL).

Далее займемся внедрением AIDL-интерфейса в служебном классе.

Внедрение AIDL-интерфейса

В предыдущем подразделе мы описали AIDL для службы, предназначенной для обработки котировок и сгенерировали файл привязки. Теперь нам предстоит внедрить эту службу в систему. Чтобы внедрить интерфейс службы, нужно написать класс, дополняющий android.app.Service и внедряющий интерфейс IStockQuoteService. Назовем класс, который мы собираемся написать, StockQuoteService. Для предоставления службы клиентам StockQuoteService должен будет обеспечить реализацию

метода onBind(). Мы добавим некоторые данные о конфигурации в файл AndroidManifest.xml. В листинге 8.12 показано, как внедряется интерфейс IStockQuoteService.

Листинг 8.12. Внедрение службы IStockQuoteService

```
// StockQuoteService.java
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;

public class StockQuoteService extends Service
{
   private static final String TAG = "StockQuoteService";
   public class StockQuoteServiceImpl extends IStockQuoteService.Stub
   {
      @Override
      public double getQuote(String ticker) throws RemoteException
      {
         Log.v(TAG, "getQuote() called for " + ticker);
         return 20.0;
      }
   }

   @Override
   public void onCreate() {
      super.onCreate();
      Log.v(TAG, "onCreate() called");
   }

   @Override
   public void onDestroy()
   {
      super.onDestroy();
      Log.v(TAG, "onDestroy() called");
   }

   @Override
   public void onStart(Intent intent, int startId) {
      super.onStart(intent, startId);
      Log.v(TAG, "onStart() called");
   }

   @Override
   public IBinder onBind(Intent intent)
   {
      Log.v(TAG, „onBind() called");
      return new StockQuoteServiceImpl();
   }
}
```

Класс StockQuoteService.java в листинге 8.12 напоминает локальный BackgroundService, созданный нами ранее, но в нем отсутствует диспетчер уведомлений. Важное отличие заключается в том, что теперь мы переходим к внедрению метода onBind(). Вспомните класс-заглушку, сгенерированный из файла AIDL, — это абстрактный класс, который внедряет интерфейс IStockQuoteService. В нашем варианте реализации службы есть внутренний класс, дополняющий класс-заглушку и называющийся StockQuoteServiceImpl. Этот класс обеспечивает внедрение удаленной службы, экземпляр данного класса возвращается методом onBind(). На этом этапе у нас уже готова рабочая AIDL-служба, но с ней пока не могут связаться внешние клиенты.

Для предоставления службы клиентам необходимо добавить объявление данной службы в файл AndroidManifest.xml. При этом нам понадобится фильтр намерений. В листинге 8.13 показано объявление службы StockQuoteService. Тег <service> является дочерним относительно тега <application>.

Листинг 8.13. Объявление службы IStockQuoteService в файле описания

```
<service android:name="StockQuoteService">
   <intent-filter>
      <action android:name=
            "com.androidbook.stockquoteservice.IStockQuoteService"/>
   </intent-filter>
</service>
```

При работе с любыми службами та служба, которую мы хотим предоставить для пользования клиентам, определяется в теге <service>. Если мы имеем дело с AIDL-службой, нам также нужно добавить тег <intent-filter> с записью <action> для интерфейса той службы, которую мы предоставляем.

После выполнения этих шагов у нас будет все необходимое для развертывания службы. Теперь рассмотрим, как вызвать службу из другого приложения (но, разумеется, с того же устройства).

Вызов службы из клиентского приложения

При обращении клиента к службе между ними должен существовать протокол или контракт. В случае с Android таким контрактом является AIDL. Итак, использование службы начинается с того, что мы берем AIDL-файл службы и копируем его в клиентский проект. При этом компилятор AIDL создает такой же файл описания интерфейса, какой создавался при внедрении службы (см. проект внедрения службы). Таким образом, клиент получает доступ ко всем методам, параметрам и типам возврата функций, используемым этой службой. Создадим новый проект и скопируем файл AIDL.

1. Создайте новый проект Android и назовите его StockQuoteClient. Выберите иное имя пакета, например com.androidbook.stockquoteclient. В поле **Create Activity** (Создание явления) укажите **MainActivity**.
2. Создайте в этом проекте новый файл Java под названием com.androidbook.stockquoteservice и расположите его в каталоге src.

3. Копируйте файл StockQuoteService.aidl из проекта StockQuoteService в этот новый проект. Обратите внимание: после копирования файла в проект компилятор AIDL сгенерирует соответствующий файл Java.

Повторно сгенерированный интерфейс службы является контрактом между клиентом и службой. На следующем этапе мы получаем ссылку на службу. Здесь мы можем вызвать метод getQuote(). При работе с удаленными службами вызывается метод bindService(), а не startService(). В листинге 8.14 показан класс явления, действующий как клиент IStockQuoteService. Кроме того, в этом листинге есть файл шаблона для явления.

Далее скопируем содержимое XML-файла из листинга 8.14 в файл /res/layout/main.xml, а содержимое файла Java — из листинга 8.14 в файл MainActivity.java. Название пакета явления не так уж и важно — явление может находиться в любом пакете на ваш выбор. Однако создаваемые вами артефакты AIDL чувствительны к названию пакета, так как компилятор AIDL генерирует код на базе содержимого AIDL-файла.

Листинг 8.14. Клиент службы IStockQuoteService

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button android:id="@+id/bindBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Bind"
   />

   <Button android:id="@+id/callBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Call Again"
   />

   <Button android:id="@+id/unbindBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="UnBind"
   />
</LinearLayout>

// это файл MainActivity.java
import com.androidbook.stockquoteservice.IStockQuoteService;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
```

```
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.os.RemoteException;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
   protected static final String TAG = "StockQuoteClient";
   private IStockQuoteService stockService = null;

   private Button bindBtn;
   private Button callBtn;
   private Button unbindBtn;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      bindBtn = (Button)findViewById(R.id.bindBtn);
      bindBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {
            bindService(new Intent(IStockQuoteService.class
                  .getName()),
                     serConn, Context.BIND_AUTO_CREATE);
            bindBtn.setEnabled(false);
            callBtn.setEnabled(true);
            unbindBtn.setEnabled(true);
         }});
      callBtn = (Button)findViewById(R.id.callBtn);
      callBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {
            callService();
         }});
      callBtn.setEnabled(false);

      unbindBtn = (Button)findViewById(R.id.unbindBtn);
      unbindBtn.setOnClickListener(new OnClickListener(){

         @Override
```

```
        public void onClick(View view) {
            unbindService(serConn);
            bindBtn.setEnabled(true);
            callBtn.setEnabled(false);
            unbindBtn.setEnabled(false);
        }});
    unbindBtn.setEnabled(false);
  }

  private void callService() {
    try {
        double val = stockService.getQuote("SYH");
        Toast.makeText(MainActivity.this, "Value from service is "+val,
                Toast.LENGTH_SHORT).show();
    } catch (RemoteException ee) {
        Log.e("MainActivity", ee.getMessage(), ee);
    }
  }
  private ServiceConnection serConn = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName name, IBinder service)
    {
        Log.v(TAG, "onServiceConnected() called");
        stockService = IStockQuoteService.Stub.asInterface(service);
        callService();
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        Log.v(TAG, "onServiceDisconnected() called");
        stockService = null;
    }
  };
}
```

Явление приводит в готовность OnClickListener для трех кнопок: **Bind** (Связать), **Call Again** (Новый вызов) и **UnBind** (Разъединить). Когда пользователь нажимает кнопку **Bind** (Связать), явление вызывает метод bindService(). Аналогично при нажатии **UnBind** (Разъединить) вызывается unbindService(). Обратите внимание: методу bindService() передается три параметра — имя AIDL-службы, экземпляр ServiceConnection и флаг для автоматического создания службы.

Имея AIDL-службу, нужно обеспечить внедрение интерфейса ServiceConnection. В этом интерфейсе определяется два метода: один вызывается системой при установлении соединения со службой, а другой — при разрыве соединения. В нашем варианте внедрения службы мы определяем закрытый член, который реализует ServiceConnection для IStockQuoteService. Когда мы вызываем метод bindService(), в нем закрытому члену передается нужная ссылка. Когда соединение со службой

установлено, мы получаем ссылку на IStockQuoteService при помощи Stub, а затем вызываем getQuote() из метода callService().

Важно отметить, что вызов bindService() является асинхронным. Поскольку служба или процесс при этом вызове может не работать, следовательно, их нужно создать или запустить. Поскольку bindService() является асинхронным, в платформе используется обратный вызов ServiceConnection, благодаря которому у нас есть информация, когда служба запускается и когда становится недоступной.

Теперь вы знаете, как создается и используется AIDL-интерфейс. Прежде чем мы перейдем дальше, к более сложным вопросам, рассмотрим, что нужно для создания простой локальной службы, с одной стороны, и AIDL-службы — с другой. Локальные службы не поддерживают onBind(). В ответ на onBind() они возвращают null. Службы такого типа доступны только для компонентов того приложения, которое является для конкретной службы несущим. Для вызова локальных служб используйте startService().

С другой стороны, AIDL-службы могут использоваться и компонентами того же процесса, в котором работает служба, и компонентами, которые действуют в других приложениях. Службы такого типа определяют в AIDL-файле контракт между собой и клиентами. Служба реализует AIDL-контракт, а клиенты связываются с AIDL-описанием. Для реализации контракта служба возвращает внедренный вариант AIDL-интерфейса из метода onBind(). Клиенты связываются с AIDL-службой, вызывая bindService(), и разрывают соединение, вызывая unbindService().

До сих пор, приводя примеры работы со службами, мы занимались только передачей простых типов Java. Но службы Android также позволяют передавать и комплексные типы. Это очень полезно, особенно при работе с AIDL-службами, поскольку у вас может быть очень длинный список параметров, которые вы хотите передать службе, и передавать их все в виде отдельных простых типов нецелесообразно. Гораздо лучше упаковать их в комплексные типы, а затем передать службе.

Рассмотрим, как происходит передача комплексных типов службам.

Передача комплексных типов службам

Выполнять передачу комплексных типов данных службам и от служб сложнее, чем передавать простые типы Java. Прежде чем взяться за эту работу, нужно усвоить, как именно в AIDL поддерживаются неэлементарные типы.

- В AIDL поддерживаются String и CharSequence.
- AIDL позволяет передавать другие AIDL-интерфейсы, но для каждого AIDL-интерфейса, на который вы ссылаетесь, в коде должен присутствовать оператор import (даже если AIDL-интерфейс, на который делается ссылка, находится в том же пакете, что и AIDL-интерфейс, из которого делается ссылка).
- В AIDL с некоторыми ограничениями поддерживаются java.util.List и java.util.Map. В коллекции могут содержаться данные следующих типов: простые типы Java, String, CharSequence или android.os.Parcelable. Для List или Map оператор Import не требуется, но такие операторы нужны для Parcelables.

- Для неэлементарных типов, кроме String, требуется индикатор направленности (directional indicator). К индикаторам направленности относятся in, out и inout. In означает, что значение устанавливается клиентом, out — значение задается службой, а inout — значение определяется как клиентом, так и службой.

Интерфейс Parcelable сообщает среде исполнения, как проводить сериализацию и десериализацию объектов в ходе процессов сборки (marshalling) и обратного преобразования (unmarshalling) аргументов. В листинге 8.15 показан класс Person, внедряющий в систему интерфейс Parcelable.

Листинг 8.15. Внедрение интерфейса Parcelable

```
// это файл Person.java
package com.syh;
import android.os.Parcel;
import android.os.Parcelable;

public class Person implements Parcelable {
   private int age;
   private String name;
   public static final Parcelable.Creator<Person> CREATOR =
              new Parcelable.Creator<Person>() {
      public Person createFromParcel(Parcel in) {
         return new Person(in);
      }

      public Person[] newArray(int size) {
         return new Person[size];
      }
   };

   public Person() {
   }

   private Person(Parcel in) {
      readFromParcel(in);
   }

   @Override
   public int describeContents() {
      return 0;
   }

   @Override
   public void writeToParcel(Parcel out, int flags) {
      out.writeInt(age);
      out.writeString(name);
   }

   public void readFromParcel(Parcel in) {
      age = in.readInt();
```

```
        name = in.readString();
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Чтобы приступить к реализации этого кода, создадим в Eclipse новый проект Android и назовем его StockQuoteService2. В поле **Create Activity** (Создание явления) укажите MainActivity и используйте пакет com.syh. Затем добавьте файл Person.java в ваш проект до пакета com.syh.

В интерфейсе Parcelable определяется контракт для наполнения объектов дополнительной информацией (hydration) и удаления такой информации (dehydration) в процессе сборки/обратного преобразования аргументов. Базой для интерфейса Parcelable является объект-контейнер Parcel. Класс Parcel служит механизмом быстрой сериализации/десериализации. Он специально разработан для обеспечения межпроцессного обмена информацией в Android. В этом классе содержатся методы, помогающие равномерно располагать элементы в контейнере либо извлекать элементы из контейнера. Чтобы правильно внедрить объект для обеспечения межпроцессного обмена информацией, сделайте следующее.

1. Внедрите интерфейс Parcelable. Под этим понимается внедрение методов writeToParcel() и readFromParcel(). Метод записи будет вносить объект в информационный пакет (parcel), а метод считывания — читать объект из этого пакета. Обратите внимание — порядок записи и порядок считывания свойств должны совпадать.
2. Добавьте свойство static final к классу с именем CREATOR. Это свойство должно внедрять интерфейс android.os.Parcelable.Creator<T>.
3. Предоставьте конструктор для Parcelable, в котором заложен алгоритм создания объекта из Parcel.
4. Определите класс Parcelable в файле AIDL, совпадающем с файлом JAVA, в котором содержится комплексный тип. Компилятор AIDL будет искать этот файл при сборке AIDL-файлов. Пример файла Person.aidl показан в листинге 8.16. Этот файл должен находиться там же, где и Person.java.

ПРИМЕЧАНИЕ

При просмотре Parcelable может возникнуть вопрос: почему в Android не применяется встроенный в Java механизм сериализации? Оказывается, что разработчики Android пришли к выводу, что сериализация в Java протекает слишком медленно и не удовлетворяет требованиям, предъявляемым в Android к межпроцессному обмену информацией. Поэтому было создано решение Parcelable. Использование Parcelable требует явной сериализации членов вашего класса, но в итоге сериализация объектов будет происходить гораздо быстрее.

Кроме того, не забудьте, что в Android существует два механизма передачи информации между процессами. Первый — это передача пакета (bundle) явлению при помощи намерения, а второй — передача Parcelable службе. Два этих механизма не взаимозаменяемы, не путайте их. Parcelable не предназначен для передачи явлению. Если вы хотите запустить явление и передать ему определенные данные, используйте пакет. Parcelable предназначен именно для использования в составе описания AIDL.

Листинг 8.16. Пример файла Person.aidl

```
package com.syh;
parcelable com.syh.Person
```

Для каждого Parcelable в вашем проекте понадобится отдельный файл AIDL. В данном случае у нас только один Parcelable — Person.

Теперь используем класс Person с удаленной службой. Для простоты изменим IStockQuoteService так, чтобы принимался входной параметр типа Person. Идея заключается в том, что клиенты будут передавать Person службе и сообщать ей, таким образом, кто запрашивает котировку. Новый вариант IStockQuoteService.aidl показан в листинге 8.17.

Листинг 8.17. Передача Parcelables службам

```
package com.syh;
import com.syh.Person;

interface IStockQuoteService
{
   String getQuote(in String ticker,in Person requester);
}
```

Теперь метод getQuote() принимает два параметра: биржевой код валюты и объект Person, указывающий, кто делает запрос. Обратите внимание: у нас есть индикаторы направленности параметров, так как параметры относятся к неэлементарным типам, и еще мы используем оператор import для класса Person. Класс Person также находится в том самом пакете, что и описание службы (com.syh).

Теперь реализация службы происходит в соответствии с листингом 8.18.

Листинг 8.18. Реализация StockQuoteService2

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="This is where the service would ask for help."
    />
</LinearLayout>

package com.syh;
// это файл StockQuoteService2.java

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.RemoteException;

public class StockQuoteService2 extends Service
{
    private NotificationManager notificationMgr;
    public class StockQuoteServiceImpl extends IStockQuoteService.Stub
    {

        @Override
        public String getQuote(String ticker, Person requester)
                throws RemoteException {
            return "Hello "+requester.getName()+"! Quote for
                        "+ticker+" is 20.0";
        }
    }

    @Override
    public void onCreate() {
        super.onCreate();

        notificationMgr =
            (NotificationManager)getSystemService(NOTIFICATION_SERVICE);

            displayNotificationMessage("onCreate()
                        called in StockQuoteService2");
    }
    @Override
    public void onDestroy()
    {
        displayNotificationMessage("onDestroy()
                        called in StockQuoteService2");
        super.onDestroy();
    }
    @Override
    public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
    }
```

```
    @Override
    public IBinder onBind(Intent intent)
    {
       displayNotificationMessage("onBind() called in StockQuoteService2");
       return new StockQuoteServiceImpl();
    }
       private void displayNotificationMessage(String message)
    {
       Notification notification = new Notification(R.drawable.note,
             message,System.currentTimeMillis());

       PendingIntent contentIntent =
             PendingIntent.getActivity(this, 0, new Intent(this,
                      MainActivity.class), 0);

       notification.setLatestEventInfo(this, "StockQuoteService2",message,
             contentIntent);

       notificationMgr.notify(R.id.app_notification_id, notification);
    }
}
```

Разница между этим и предыдущим вариантами заключается в том, что мы вернули уведомления (notifications), и в том, что теперь возвращаем значение котировки в виде строки, а не в виде числа с плавающей запятой.

Строка, возвращаемая пользователю, содержит имя запрашивающего лица из объекта Person. На этом примере видно, что мы считали значение, посланное от клиента, и что объект Person был правильно передан службе.

Для выполнения задачи нужно сделать еще несколько шагов.

1. Добавить файл изображения note в каталог /res/drawable.
2. Добавить новый тег <item type="id" name="app_notification_id"/> в файл /res/values/strings.xml.
3. Изменить файл AndroidManifest.xml приложения в соответствии с листингом 8.19.

Листинг 8.19. Измененный тег <application> в файле AndroidManifest.xml для StockQuoteService2

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
   <activity android:name=".MainActivity"
         android:label="@string/app_name">
      <intent-filter>
         <action android:name="android.intent.action.MAIN" />
      </intent-filter>
   </activity>
<service android:name="StockQuoteService2">
   <intent-filter>
      <action android:name="com.syh.IStockQuoteService" />
   </intent-filter>
</service>
</application>
```

Наконец, мы воспользуемся стандартным файлом MainActivity.java, который просто отображает базовый шаблон с простым сообщением. Теперь, когда наша служба внедрена, создадим новый проект Android под названием StockQuoteClient2. Названием пакета будет com.sayed, а названием явления — MainActivity. Для внедрения клиента, который будет передавать объект Person службе, нужно скопировать всю информацию, необходимую клиенту, из проекта службы в проект клиента. В предыдущем примере нам понадобился только файл IStockQuoteService.aidl. Теперь нужно скопировать два файла — Person.java и Person.aidl, поскольку в данном случае объект Person входит в состав интерфейса. Скопировав в клиентский проект три файла, измените main.xml и MainActivity.java в соответствии с листингом 8.20.

Листинг 8.20. Вызов службы с Parcelable

```
<?xml version="1.0" encoding="utf-8"?>
<!--Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button android:id="@+id/bindBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Bind"
   />
   <Button android:id="@+id/callBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Call Again"
   />

   <Button android:id="@+id/unbindBtn"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="UnBind"
   />
</LinearLayout>

package com.sayed;
// это файл MainActivity.java

import com.syh.IStockQuoteService;
import com.syh.Person;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
```

```
import android.os.RemoteException;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {

   protected static final String TAG = "StockQuoteClient2";
   private IStockQuoteService stockService = null;

   private Button bindBtn = null;
   private Button callBtn = null;
   private Button unbindBtn = null;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      bindBtn = (Button)findViewById(R.id.bindBtn);
      bindBtn.setOnClickListener(new OnClickListener(){

      @Override
      public void onClick(View view) {
         bindService(new Intent(IStockQuoteService.class
               .getName()),
               serConn, Context.BIND_AUTO_CREATE);
         bindBtn.setEnabled(false);
         callBtn.setEnabled(true);
         unbindBtn.setEnabled(true);
      }});

   callBtn = (Button)findViewById(R.id.callBtn);
   callBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {
            callService();
         }});
      callBtn.setEnabled(false);

      unbindBtn = (Button)findViewById(R.id.unbindBtn);
      unbindBtn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {
            unbindService(serConn);
            bindBtn.setEnabled(true);
            callBtn.setEnabled(false);
```

```
            unbindBtn.setEnabled(false);
        }});
        unbindBtn.setEnabled(false);
    }

    private void callService() {
        try {
            Person person = new Person();
            person.setAge(33);
            person.setName("Sayed");
            String response = stockService.getQuote("GOOG", person);
            Toast.makeText(MainActivity.this, "Value from service is
                        "+response,Toast.LENGTH_SHORT).show();
        } catch (RemoteException ee) {
            Log.e("MainActivity", ee.getMessage(), ee);
        }
    }
    private ServiceConnection serConn = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName name, IBinder service)
        {
            Log.v(TAG, "onServiceConnected() called");
            stockService = IStockQuoteService.Stub.asInterface(service);
            callService();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            Log.v(TAG, "onServiceDisconnected() called");
            stockService = null;
        }
    };
}
```

Теперь все должно заработать. Не забудьте сначала передать службу в эмулятор, а потом клиенту для запуска. Посмотрим, что у нас получилось. При помощи метода onServiceConnected() мы узнаем, что наша служба запущена, поэтому вызываем метод callService(). Далее создаем новый объект Person, устанавливаем для него свойства Age и Name. Затем выполняем службу и отображаем результат вызова службы. На рис. 8.1 показано, что у нас получится.



Рис. 8.1. Результат вызова функции с Parcelable

Обратите внимание, что при вызове службы в строке состояния выводится соответствующее уведомление. Оно приходит от самой службы. Мы коротко поговорили об уведомлениях выше, когда рассматривали их в контексте обмена ин-

формацией между службой и пользователем. Как правило, службы работают в фоновом режиме и не имеют никакого пользовательского интерфейса. Но что делать, если пользователь должен взаимодействовать с той или иной службой? Напрашивается мысль о том, что служба могла бы активировать для этого специальное явление, но служба *ни в коем случае* не должна активировать явления напрямую. Вместо этого она должна выдавать уведомление, в котором будет сообщаться, как получить нужное явление. Этот принцип был проиллюстрирован в нашем последнем упражнении. Мы определили простой шаблон и явление, внедренное специально для работы с нашей службой. Когда мы создавали в службе уведомление, то сделали так, чтобы оно включало в себя явление. Пользователь нажимает уведомление — после этого открывается явление, входящее в состав нашей службы. Так обеспечивается взаимодействие между пользователем и службой.

Уведомления сохраняются, поэтому, чтобы просмотреть их, можно открыть меню на главной странице Android и нажать **Notifications** (Уведомления). Пользователь может перетаскивать пиктограммы уведомлений в строку состояния, чтобы они были видны в любой момент. Обратите внимание — мы используем вызов метода `setLatestEventInfo()`. А это значит, что многократно применяем один и тот же ID с каждым сообщением. Такая комбинация позволяет нам всякий раз обновлять (и изменять) единственный экземпляр уведомления, а не создавать все новые записи уведомлений. Следовательно, при переходе в окно уведомлений Android после нескольких нажатий **Bind** (Связать), **Call Again** (Новый вызов) и **Unbind** (Разъединить) вы увидите в разделе уведомлений только одно сообщение, а именно то, которое было в последний раз послано `BackgroundService`. Если бы мы использовали разные ID, у нас могло бы быть несколько уведомлений и мы могли бы обновлять каждое из них отдельно. Пользователь также может устанавливать для уведомлений дополнительные «подсказки» — звук, свет и/или вибрацию.

Кроме того, будет полезно просмотреть артефакты, входящие в состав проекта службы, и клиента, который вызывает службу (рис. 8.2).

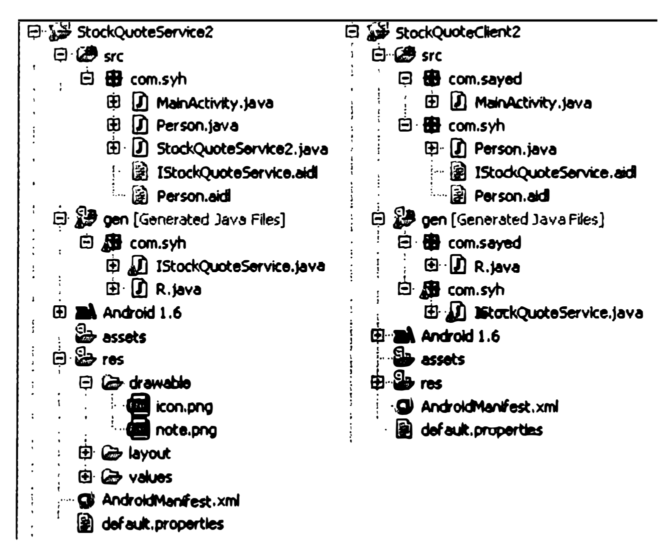


Рис. 8.2. Артефакты службы и клиента

На рис. 8.2 показаны артефакты проекта Eclipse, относящиеся к службе *(слева)* и к клиенту *(справа)*. Обратите внимание на то, что контракт между клиентом и службой состоит из артефактов AIDL и объектов `Parceable`, которыми обмениваются две стороны. Именно поэтому `IStockQuoteService.aidl`, `Person.java` и `Person.aidl` есть с обеих сторон. Поскольку AIDL-компилятор генерирует интерфейс Java, заглушку, прокси и т. д. из артефактов AIDL, процесс сборки создает файл `IStockQuoteService.java` с клиентской стороны, копируя артефакты контракта в клиентский проект.

Теперь вы знаете, как организуется обмен комплексными типами между службами и клиентами. Кратко рассмотрим еще один аспект вызова служб: сравним синхронные и асинхронные вызовы.

Все вызовы служб, которые вы выполняете, являются синхронными. Возникает вопрос — нужно ли внедрять все служебные вызовы в рабочем потоке (worker thread)? Нет, это не обязательно. На большинстве других платформ распространены ситуации, когда клиент пользуется службами, которые являются настоящими «черными ящиками» (то есть их внутренняя структура неизвестна), поэтому при выполнении служебных вызовов клиент должен соблюдать определенные меры предосторожности. В Android структура службы обычно известна (потому что, как правило, вы сами пишете службу), поэтому вам не придется решать вслепую. Если вы знаете, что вызываемый метод должен выполнять много сложной работы, подумайте, можно ли воспользоваться при вызове дополнительным вторичным потоком. Если вы уверены, что метод не доставит никаких проблем, можете смело делать вызов в потоке пользовательского интерфейса. Если вы решите, что лучше сделать вызов в рабочем потоке, можете создать поток при помощи метода `onServiceConnected()`, относящегося к `ServiceConnection`, а затем вызвать службу. Полученный результат можно после этого передать в поток пользовательского интерфейса.

Резюме

В этой главе мы говорили о службах. Мы обсудили вопросы использования внешних HTTP-служб при помощи Apache `HttpClient` и поговорили о написании фоновых служб. Что касается применения `HttpClient`, мы показали, как выполняются вызовы HTTP `GET` и HTTP `POST`. Мы также объяснили, как вызовы POST могут включать в себя несколько частей.

Во второй части главы рассматривались вопросы, связанные с написанием служб в Android. В частности, вы научились писать локальные и удаленные службы. Мы рассказали, что локальные службы используются компонентами (например, явлениями) того же процесса, в котором была запущена служба. Клиенты удаленных процессов работают вне процесса, являющегося для службы несущим (hosting).

В следующей главе мы поговорим о поддержке телефонии и мультимедийных функций в Android.

9 Использование медиафреймворка и интерфейсов API для функций телефонии

В этой главе мы исследуем два очень интересных раздела Android SDK: медийные компоненты и компоненты для работы с телефонией. В первой части главы мы поговорим о медиа, покажем, как воспроизводить и записывать аудио и видео. Во второй части главы речь пойдет о телефонии. Мы рассмотрим, как отправлять и получать короткие сообщения (SMS). Кроме того, затронем некоторые интересные аспекты, касающиеся телефонии в Android.

Начнем с рассмотрения медийных API.

Использование медийных API-интерфейсов

В Android поддерживается воспроизведение аудио- и видеоконтента. Для этого применяются функции из пакета `android.media`. В этом разделе мы рассмотрим медийные API, входящие в состав данного пакета.

Основной составляющей пакета `android.media` является класс `android.media.MediaPlayer`. Класс `MediaPlayer` отвечает за воспроизведение как аудио, так и видео. Контент для этого класса может приходить из следующих источников.

- *Веб* — можно воспроизводить контент из веб по гиперссылке.
- *APK-файл* — можно воспроизводить контент, упакованный в APK-файл. Контент можно упаковать как ресурс или как актив (в каталоге `assets`).
- *Карта памяти* — можно воспроизводить контент, находящийся на карте памяти, вставленной в устройство.

Медиаплеер может декодировать различные форматы контента, в том числе 3GPP, MP3, MIDI, PCM/WAVE и MPEG-4. Полный список поддерживаемых форматов медиа приведен на странице по адресу **http://developer.android.com/guide/appendix/media-formats.html**.

Карты памяти

Прежде чем перейти к созданию и использованию различных типов медиа, научимся работать с картами памяти, одним из источников контента медиаплеера. Карты памяти используются в телефонах Android для сохранения больших объемов пользовательских данных, обычно медийного плана, в частности изображений, аудио или видео. Карты памяти — это чаще всего подключаемые микросхемы памяти, данные на которых сохраняются даже при отключении питания. В телефоне имеется специальное гнездо, в которое вставляется карта памяти. Информацию с нее можно просматривать на устройстве. К одному устройству можно подключать несколько карт памяти, а потом переходить между ними, а также использовать карты памяти в разных устройствах. Очень удобно, что эмулятор Android может имитировать работу с картами памяти, используя пространство вашего жесткого диска так, как если бы он был подключаемой картой памяти.

Когда вы впервые создавали виртуальное устройство Android (AVD) в главе 2, то указывали объем карты памяти, доступный вашей программе при запуске в эмуляторе. Если посмотреть содержимое каталога AVD, то в нем окажется файл sdcard.img, имеющий размер, который вы указали. Тогда мы не пользовались картой памяти, но будем работать с ней в этой главе. Занимаясь разработкой программ, вы можете использовать карту памяти и сохранять на ней медийные (или другие) файлы при помощи специальных инструментов Android, доступных в Eclipse. Кроме того, можно воспользоваться утилитой adb (Android Debug Bridge) для сохранения файлов на карте памяти и взятия их оттуда. Утилита adb находится в подкаталоге tools инструментария Android SDK. Туда можно попасть из окна инструментов (как это сделать, описано в главе 2).

Вы уже знаете, как получить карту памяти, создавая AVD. И, разумеется, можно создать сколько угодно AVD, не отличающихся ничем, кроме размера используемых в них карт памяти. Но можно поступить иначе. В комплекте инструментов Android SDK есть утилита, которая называется mksdcard и может создавать образ карты памяти. Если быть точными, эта утилита создает отформатированный файл, используемый в качестве карты памяти. Чтобы работать с этой утилитой, сначала создайте каталог для файла образа, например c:\Android\sdcard\. Затем откройте окно инструментов и запустите команду следующего вида, указав верный путь к файлу образа карты памяти:

```
mksdcard 256M c:\Android\sdcard\sdcard.img
```

В этом примере команда создает образ файла по адресу c:\Android\sdcard\, имя этого файла — sdcard.img. Карта памяти будет иметь размер 256 Мбайт. Для указания других размеров можно использовать килобайты (K), а вот гигабайты (G) в данном случае указывать нельзя. Поэтому, чтобы получить гигабайтные величины, мегабайты умножаются на 1024. Можно просто указать целое число, соответствующее общему количеству байт. Обратите также внимание на то, что эмулятор Android не работает с картами памяти размером менее 8 Мбайт.

Набор инструментов для разработки в Android (ADT) в Eclipse позволяет указывать дополнительные аргументы командной строки при запуске эмулятора. Чтобы найти поле для задания параметров эмулятора, перейдите в окно Preferences

(Настройки) в Eclipse, затем выберите **Android ▸ Launch** (Android ▸ Запуск). Теоретически здесь можно указать `-sdcard "PATH_TO_YOUR_SD_CARD_IMAGE_FILE` (путь к вашей карте памяти) и переопределить таким образом путь к карте памяти, заданный в виртуальном устройстве. Но пока такой метод не работает в некоторых релизах Android, и вы всегда будете получать тот образ карты памяти, который создавался вместе с AVD. Если с AVD используется отдельная карта памяти, лучше всего запускать эмулятор из командной строки и указывать образ карты памяти для этой цели. Если работать с окном инструментов, то приведенная ниже команда запускает указанное AVD, но использует карту памяти, созданную вами, а не ту, которая создавалась вместе с этим AVD:

```
emulator -avd AVDName -sdcard "PATH_TO_YOUR_SD_CARD_IMAGE_FILE"
```

Когда карта памяти только создается, на ней еще нет никаких файлов. Чтобы переместить на нее файлы, используйте инструмент **File Explorer** (Проводник по файлам), входящий в состав Eclipse. Запустите эмулятор и подождите, пока он инициализируется. Затем выберите в Eclipse один из режимов — **Java**, **Debug** или **DDMS**, а затем откройте вкладку **File Explorer** (Проводник по файлам) (рис. 9.1).

Рис. 9.1. Вид File Explorer

Если **File Explorer** (Проводник по файлам) не отображается, выполните команду **Window ▸ Show View ▸ Other ▸ Android** (Окно ▸ Показать вид ▸ Другие ▸ Android) и выберите **File Explorer** (Проводник по файлам). Вы можете также перейти в режим **Dalvik Debug Monitor Service (DDMS)** следующим образом: **Window ▸ Open Perspective ▸ Other ▸ DDMS** (Окно ▸ Открыть режим ▸ Другие ▸ DDMS), отобразив, таким образом, все имеющиеся виды (рис. 9.2).

Чтобы переместить файл на карту памяти, выберите в **File Explorer** (Проводник по файлам) каталог `sdcard` и найдите кнопку со стрелкой вправо (в верхнем правом углу), указывающую на схематичное изображение телефона. Так запускается диалоговое окно, в котором можно выбрать файл. Выберите файл, который хотите перенести на карту памяти. Далее идет кнопка, на которой нарисована стрелка, указывающая на дискету. При помощи этой кнопки можно перетащить файл с устройства на рабочую станцию (предварительно нужно выбрать файл для перетаскивания в **File Explorer** (Проводник по файлам)).

Если в **File Explorer** (Проводник по файлам) отображается пустое поле, причин может быть несколько: у вас не работает эмулятор, разорвано соединение между Eclipse и эмулятором либо проект, который работает у вас в эмуляторе, не выбран на вкладке **Devices** (Устройства).

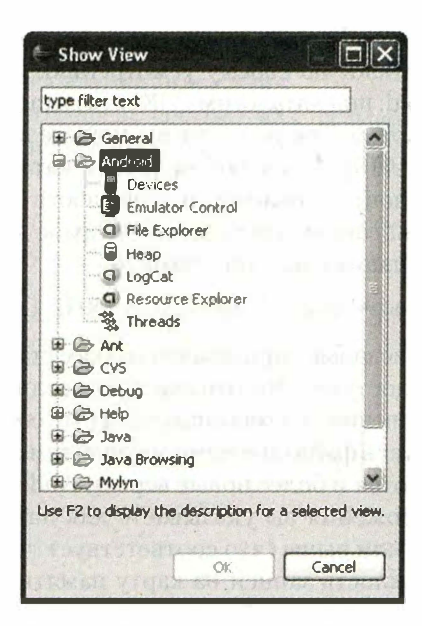


Рис. 9.2. Включение видов в Android

Еще один способ переноса файлов на карту памяти и с нее на другой носитель — применение утилиты adb. Чтобы использовать этот способ, откройте окно инструментов, а затем наберите команду следующего вида:

```
adb push c:\path_to_my_file\filename /sdcard/newfile
```

В результате выбранный файл переместится с рабочей станции на карту памяти. Обратите внимание: в качестве разделительного знака между каталогами на устройстве применяются прямые слэши. Используйте такой разделительный знак, который принят при работе на вашей рабочей станции, а также правильно указывайте путь при перемещении файла. Далее приведена обратная команда, позволяющая переместить файл с карты памяти на рабочую станцию:

```
adb pull /sdcard/devicefile c:\path_to_where_its_going\filename
```

Одна из приятных черт этой команды заключается в том, что она создает каталоги по мере надобности в любом направлении (перемещение на карту памяти или с нее), чтобы файл попал именно туда, куда нужно. К сожалению, adb не позволяет одновременно копировать несколько файлов. Каждый файл переносится отдельно.

Наверное, вы заметили, что на карте памяти есть каталог, называемый DCIM. Здесь хранятся изображения с цифрового фотоаппарата (**D**igital **C**amera **Im**ages). Существует промышленный стандарт, в соответствии с которым каталог DCIM помещается в корневой каталог карты памяти, используемый для хранения цифровых изображений. В соответствии с еще одним промышленным стандартом под каталогом DCIM создается каталог с именем в формате 123ABCDE (три цифры, за ними пять букв), представляющий конкретную камеру. Эмулятор создает под DCIM каталог

100ANDRO, но производители цифровых камер и телефонов, работающих с Android, переименовывают этот каталог по своему усмотрению. В эмуляторе, как и в некоторых телефонах Android, под каталогом DCIM есть каталог Camera, но это не соответствует никаким стандартам. Тем не менее вы можете встретить файлы изображений и под Camera, и под 100ANDRO, и в любом другом каталоге под DCIM.

И наконец, несколько слов о безопасности. При работе с Android SDK версии 1.6 необходимо добавить в файл описания следующее право доступа, чтобы ваша программа могла записывать данные на карту памяти:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Однако программы, написанные с применением более старых версий Android SDK, не требуют такого права доступа. Это означает, что если параметр minSdkVersion вашей программы имеет значение, не превышающее 4 (что соответствует Android SDK версии 1.6), то указанный тег в файл описания добавлять не нужно, даже если вашим устройством поддерживаются и более новые версии Android SDK. В дальнейшем, если при написании приложения вы указываете для параметра **Build Target** (Построить цель) Android 1.6 или выше (что соответствует minSdkVersion 4 или выше) и хотите включить возможность записи на карту памяти, убедитесь, что в файле описания проставлен вышеуказанный тег. Если в роли цели (Target) выступает Android 1.5 или выше, этот тег нужен. Теперь, когда вы знаете основы работы с картами памяти, перейдем к работе с аудио.

Воспроизведение аудио

Для начала напишем простую программу, которая позволяет воспроизводить MP3-файлы, расположенные в Интернете (рис. 9.3). После этого поговорим об использовании метода setDataSource() из класса MediaPlayer. Этот метод позволяет воспроизводить контент из файла APK или с карты памяти. В заключение разговора о медиа обсудим некоторые недостатки медийных API.



Рис. 9.3. Пользовательский интерфейс программы для работы с медиафайлами

На рис. 9.3 показан пользовательский интерфейс для нашего первого примера. В этой программе мы изучим некоторые фундаментальные аспекты, связанные с применением класса MediaPlayer. В частности, научимся запускать воспроизведе-

ние медиафайла, приостанавливать его и перезапускать. Рассмотрим шаблон пользовательского интерфейса этого приложения.

Пользовательский интерфейс состоит из LinearLayout с тремя кнопками: для запуска плеера, паузы и перезапуска файла. Код для этого файла и его шаблон показаны в листинге 9.1.

Листинг 9.1. Код и шаблон программы для работы с медиа

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<Button android:id="@+id/startPlayerBtn"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Start Playing Audio"
   />

<Button android:id="@+id/restartPlayerBtn"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Restart Player"
   />

<Button android:id="@+id/pausePlayerBtn"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Pause Player"
   />
</LinearLayout>

import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity
{
   static final String AUDIO_PATH =
         "http://www.androidbook.com/akc/filestorage/android
                              /documentfiles/3389/play.mp3";

private MediaPlayer mediaPlayer;
private int playbackPosition=0;

/** Вызывается при первом создании явления. */
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    Button startPlayerBtn = (Button)findViewById(R.id.startPlayerBtn);
    Button pausePlayerBtn = (Button)findViewById(R.id.pausePlayerBtn);
    Button restartPlayerBtn = (Button)findViewById(R.id.restartPlayerBtn);

    startPlayerBtn.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View view)
        {
            try {
                playAudio(AUDIO_PATH);
                // playLocalAudio();
                // playLocalAudio_UsingDescriptor();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }});

    pausePlayerBtn.setOnClickListener(new OnClickListener(){
        @Override
        public void onClick(View view)
        {
            if(mediaPlayer!=null)
            {
                playbackPosition = mediaPlayer.getCurrentPosition();
                mediaPlayer.pause();
            }
        }});

    restartPlayerBtn.setOnClickListener(new OnClickListener(){

        @Override
        public void onClick(View view)
        {
            if(mediaPlayer!=null && !mediaPlayer.isPlaying())
            {
                mediaPlayer.seekTo(playbackPosition);
                mediaPlayer.start();
            }
        }});
}

private void playAudio(String url)throws Exception
{
    killMediaPlayer();

    mediaPlayer = new MediaPlayer();
    mediaPlayer.setDataSource(url);
    mediaPlayer.prepare();
```

```
        mediaPlayer.start();
    }

    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        killMediaPlayer();
    }
    private void killMediaPlayer()
    {
        if(mediaPlayer!=null)
        {
            try
            {
                mediaPlayer.release();
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

В этом сценарии MP3-файл воспроизводится с веб-адреса. Следовательно, в файл описания нужно добавить право доступа android.permission.INTERNET. Код из листинга 9.1 показывает, что в классе MainActivity содержится три члена: строка final, которая указывает URL MP3-файла, экземпляр MediaPlayer и целое число, называемое playbackPosition. По методу onCreate() понятно, что код приводит в состояние готовности слушатели щелчков для трех кнопок. При обработке нажатия кнопки **Playing Audio** (Воспроизведение аудио) вызывается метод playAudio(). В методе playAudio() создается новый экземпляр MediaPlayer, а в качестве источника данных для плеера задается URL MP3-файла. Затем вызывается другой метод плеера, prepare(), подготавливающий плеер для воспроизведения контента, и, наконец, запускается метод start(), который запускает воспроизведение.

Теперь рассмотрим обработчики нажатий кнопок Pause Player и Restart Player. Как видите, когда выбрана кнопка Pause Player, актуальное положение плеера можно узнать, вызвав метод getCurrentPosition(). Чтобы приостановить работу плеера, вызовите метод pause(). Чтобы возобновить работу плеера, нужно вызвать метод seekTo(), передать ему положение, взятое из метода getCurrentPosition(), а потом вызвать start().

В классе MediaPlayer также содержится метод stop(). Обратите внимание, что при остановке плеера методом stop() перед следующим вызовом start() нужно сначала вызвать prepare(). В свою очередь, если вызвать pause(), то start() можно вызывать сразу же, не подготавливая плеер к воспроизведению. Кроме того, не забывайте вызывать метод release(), когда закончите работу с плеером. В данном примере эта операция выполняется в рамках метода killMediaPlayer().

Итак, в листинге 9.1 показано, как воспроизвести аудиофайл, находящийся в Интернете. Класс MediaPlayer также позволяет воспроизводить и локальные

медиафайлы. Листинг 9.2 показывает, как поставить ссылку на файл, находящийся в каталоге /res/raw, и воспроизвести его. После этого в проекте Eclipse добавьте каталог raw в /res, если он находится еще не там. Затем скопируйте выбранный вами MP3-файл в /res/raw под именем music_file.mp3.

Листинг 9.2. Использование MediaPlayer для воспроизведения файла с того же устройства, на котором расположена программа

```
private void playLocalAudio()throws Exception
{
   mediaPlayer = MediaPlayer.create(this, R.raw.music_file);
   mediaPlayer.start();
}
```

Если вам нужно включить в программу аудио- или видеофайл, этот файл нужно поместить в каталог /res/raw. Затем можно получить экземпляр MediaPlayer для ресурса, передав ID ресурса, присвоенный медиафайлу. Это делается при помощи статического метода create() (см. листинг 9.2). Обратите внимание: в классе MediaPlayer также имеются статические методы create(), которые можно использовать для получения MediaPlayer, чтобы не приходилось инстанцировать новый экземпляр этого класса. Например, в листинге 9.2 создается метод create(), но вместо этого можно вызвать конструктор MediaPlayer(Context context,int resourceId). Рекомендуется пользоваться статическими методами create(), так как они скрывают акт создания MediaPlayer. Однако ниже будут показаны случаи, в которых вы не можете выбрать из двух возможностей — вы должны инстанцировать заданный по умолчанию конструктор, так как медийный контент нельзя найти по ID ресурса или по URL.

Метод setDataSource

В листинге 9.2 мы вызывали метод create() для загрузки файла с необработанного ресурса. При использовании такого метода не требуется вызывать метод setDataSource(). В качестве альтернативы можно самостоятельно инстанцировать MediaPlayer при помощи конструктора, применяемого по умолчанию, либо если ваш медийный контент нельзя получить по ID ресурса или по URL, нужно вызвать метод setDataSource().

В методе setDataSource() есть замещаемые версии, которые можно использовать для специальной настройки источника данных в соответствии с вашими нуждами. Например, в листинге 9.3 показано, как загрузить аудиофайл с необработанного ресурса при помощи FileDescriptor.

Листинг 9.3. Определение источника данных для медиаплеера с применением FileDescriptor

```
private void playLocalAudio_UsingDescriptor() throws Exception {

   AssetFileDescriptor fileDesc = getResources().openRawResourceFd(
         R.raw.music_file);
   if (fileDesc != null) {

      mediaPlayer = new MediaPlayer();
```

```
        mediaPlayer.setDataSource(fileDesc.getFileDescriptor(), fileDesc
              .getStartOffset(), fileDesc.getLength());

        fileDesc.close();

        mediaPlayer.prepare();
        mediaPlayer.start();
    }
}
```

Код, приведенный в листинге 9.3, должен находиться в контексте явления. Как видите, для получения ресурсов приложения вызывается метод getResources(), а потом применяется метод penRawResourceFd(), при помощи которого вы получаете дескриптор для аудиофайла из каталога /res/raw. Затем вызывается метод setDataSource(). При этом применяется AssetFileDescriptor, начальная позиция, с которой начинается воспроизведение файла, и конечная позиция. Такой вариант setDataSource() можно использовать и в тех случаях, когда требуется воспроизвести определенный фрагмент аудиофайла. Если вы всегда хотите воспроизводить целый файл, вызывается упрощенная версия setDataSource(FileDescriptor desc), не требующая указания первичного отступа и длины.

По этой причине может быть удобно использовать один из методов setDataSource() с FileDescriptor в тех случаях, когда вам требуется воспроизвести медийный файл, расположенный в каталоге /data вашей программы. Из соображений безопасности, медиаплеер не имеет доступа к каталогу программы /data, но программа может открыть определенный файл, а затем передать (открытый) FileDescriptor методу setDataSource(). Обратите внимание: каталог /data вашей программы находится среди файлов и каталогов, расположенных под /data/data/*APP_PACKAGE_NAME*/. Вы можете получить доступ к этому каталогу, вызвав нужный метод класса Context, то есть можно не задавать путь к каталогу в коде. Например, можно применить getFilesDir() к Context, чтобы узнать путь к каталогу файлов действующей программы. В данном случае путь будет таким: /data/data/*APP_PACKAGE_NAME*/files. Аналогично при помощи getCacheDir() можно получить путь к каталогу кэша программы. Программа будет иметь право чтения и изменения содержимого этих каталогов, поэтому файлы можно создавать динамически и подавать их на плеер. Наконец, если вы используете FileDescriptor так, как это показано в листинге 9.3, обязательно закрывайте дескриптор после вызова setDataSource().

Обращаем ваше внимание на то, что каталог /data вашей программы серьезно отличается от его же каталога /res/raw. Физически каталог /res/raw является частью архива APK. При этом данный каталог статичен. Это означает, что нельзя динамически вносить изменения в файл APK. Содержимое каталога /data, в свою очередь, является динамическим.

Нужно поговорить еще об одном источнике, с которого могут воспроизводиться аудиофайлы: о карте памяти. Выше мы показали, как переместить информацию на карту памяти. Использовать карты памяти с MediaPlayer очень просто. В предыдущем примере мы применяли setDataSource() для доступа к информации, находящейся в Интернете; при этом в методе передавалась URL MP3-файла. Если у вас на карте памяти есть аудиофайл, можно применять тот же метод setDataSource(), но вместо URL в методе передается путь к аудиофайлу, находящемуся на карте

памяти. Например, если поместить MP3-файл music_file.mp3 в каталог /sdcard, то переменную AUDIO_PATH можно преобразовать так: /sdcard/music_file.mp3. Файл будет воспроизводиться следующим образом:

```
static final String AUDIO_PATH = "/sdcard/music_file.mp3";
```

На этом мы завершаем разговор о воспроизведении аудио. Теперь обратимся к работе с видео. Вы увидите, что механизм выставления ссылок на видеоконтент очень похож на соответствующий механизм выставления ссылок на аудиофайлы.

Воспроизведение видео

В этом подразделе будет рассмотрено воспроизведение видео при помощи средств, входящих в состав Android SDK. В частности, мы поговорим о воспроизведении видео с веб-сервера и с карты памяти. Несложно догадаться, что проигрывание видео — это более сложный процесс, чем воспроизведение аудио. К счастью, в Android SDK имеются дополнительные абстракции, выполняющие большую часть работы, связанной с этим процессом.

Итак, воспроизводить видео сложнее, чем аудио, так как обрабатывается не только звук, но и видеоинформация. Чтобы облегчить жизнь разработчику, Android предоставляет специальный элемент управления просмотром android.widget.VideoView, который отвечает за создание и инициализацию MediaPlayer. Для воспроизведения видео создается виджет VideoView, который в дальнейшем используется в качестве содержимого пользовательского интерфейса. Затем задается путь к видеофайлу или URI этого файла и запускается метод start(). В листинге 9.4 показано, как в Android воспроизводится видео.

Листинг 9.4. Воспроизведение видео с применением медийного API

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout
   android:layout_width="fill_parent" android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android">

      <VideoView
         android:id="@+id/videoView"
         android:layout_width="200px"
         android:layout_height="200px" />

</LinearLayout>

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity {
   /** Вызывается при первом создании явления. */
   @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.main);

        VideoView videoView = (VideoView)this.findViewById(R.id.videoView);
        MediaController mc = new MediaController(this);
        videoView.setMediaController(mc);
        videoView.setVideoURI(Uri.parse(
                "http://www.androidbook.com/akc/filestorage/android/
                    documentfiles/3389/movie.mp4"));
        // videoView.setVideoPath("/sdcard/movie.mp4");
        videoView.requestFocus();
        videoView.start();
    }
}
```

В примере из листинга 9.4 показано, как воспроизвести видеофайл, расположенный в Интернете по адресу http://www.androidbook.com/akc/filestorage/android/documentfiles/3389/movie.mp4. Это означает, что приложение, выполняющее код, должно иметь право доступа android.permission.INTERNET. Вся функциональность, обеспечивающая воспроизведение видео, находится в классе VideoView. На самом деле все, что от вас требуется, — подать видео в видеоплеер. Пользовательский интерфейс этой программы показан на рис. 9.4.

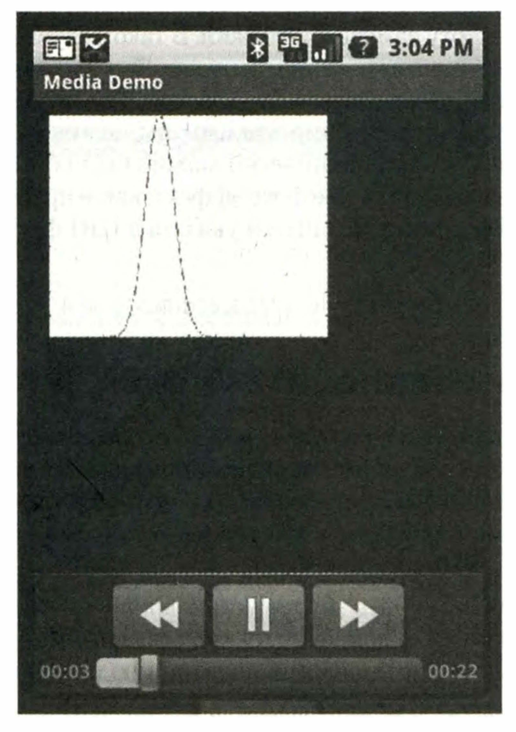


Рис. 9.4. Пользовательский интерфейс для воспроизведения видео с включенными элементами управления медиа

Работая с этой программой, вы увидите в нижней части экрана кнопки, которые появятся примерно на три секунды, а потом исчезнут. Чтобы они снова появились, нужно щелкнуть в любой точке видеокадра. При воспроизведении аудио нужно отображать только те кнопки, которые предназначены для запуска, остановки и перезапуска аудио. Для аудио нам не требуется само окно просмотра. При работе с видео, конечно же, нужны и кнопки для управления записью, и экран для просмотра. В данном примере используем для отображения видео компонент VideoView. Но мы можем не создавать кнопки самостоятельно (хотя, в принципе, это возможно), а создадим MediaController, который предоставит нам готовые кнопки. На рис. 9.4 и в листинге 9.4 показано, как настроить медиаконтроллер для VideoView путем вызова setMediaController(), который активирует элементы управления для воспроизведения, паузы и перехода к определенному фрагменту. Если вы хотите запрограммировать управление видео, используя собственные кнопки, можете вызвать методы start(), pause(), stopPlayback() и seekTo().

Не забывайте, что в этом примере мы по-прежнему пользуемся MediaPlayer — его просто не видно. На самом деле видео можно воспроизводить прямо в MediaPlayer. Вы можете вернуться к примеру, рассмотренному в листинге 9.1, записать на карту памяти видеоролик и вставить вместо AUDIO_PATH путь к видеофайлу. И вы увидите, что плеер очень хорошо воспроизводит аудиодорожку, хотя видео и не отображается.

В MediaPlayer есть метод setDataSource(), а в VideoView — нет. В VideoView вместо setDataSource() применяются методы setVideoPath() или setVideoURI(). Предположим, вы сохраняете на карту памяти видеофайл. В таком случае в коде из листинга 9.4 вызов setVideoURI() оформляется как комментарий, а вызов setVideoPath(), в свою очередь, перестает быть комментарием. Кроме того, укажите правильный путь к видеофайлу. При повторном запуске программы вы не только *увидите* в VideoView *картинку*, но и услышите *звук*. Технически можно было бы вызвать setVideoURI() со следующим кодом и получить такой же эффект, как и при вызове setVideoPath(). Но при этом нужно быть внимательным и указать в URI файла три прямых слэша между file: и sdcard:

```
videoView.setVideoURI(Uri.parse("file:///sdcard/movie.mp4"));
```

Характерные особенности MediaPlayer

В целом MediaPlayer построен в соответствии со строгой системой, то есть необходимо вызывать операции в заданном порядке, чтобы инициализация плеера прошла правильно и программа была подготовлена к воспроизведению данных. В следующем списке обобщены некоторые характерные особенности, связанные с использованием медийного API.

- После того как для MediaPlayer будет установлен источник данных, его нельзя просто заменить другим — для этого нужно создать новый MediaPlayer либо вызвать метод reset(), чтобы произвести повторную инициализацию плеера.
- После вызова prepare() можно вызвать getCurrentPosition() getDuration() и isPlaying(), чтобы узнать текущее состояние плеера. Кроме того, после prepare() можно вызвать методы setLooping() и setVolume().

- После вызова start() можно вызвать pause(), stop() и seekTo().
- Каждый MediaPlayer создает новый поток, поэтому, закончив работу с MediaPlayer, обязательно вызывайте метод release(). При воспроизведении видео с помощью VideoView работа заканчивается автоматически, но при использовании MediaPlayer работу нужно завершать вручную.

Теперь изучим, как записывать медиаинформацию.

Изучение аудиозаписи

В медиафреймворке Android поддерживается запись аудио. Аудио записывается при помощи класса android.media.MediaRecorder. В этом разделе мы покажем, как создать программу, которая сначала записывает аудиоконтент, а потом воспроизводит его. Пользовательский интерфейс этой программы показан на рис. 9.5.



Рис. 9.5. Пользовательский интерфейс программы для записи аудио

На рис. 9.5 видно, что интерфейс программы содержит четыре кнопки: две для записи, две — для запуска и остановки воспроизведения записанного контента. В листинге 9.5 показан файл шаблона и класс явления для пользовательского интерфейса.

Листинг 9.5. Запись и воспроизведение медиа в Android

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/record.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <Button android:id="@+id/bgnBtn" android:layout_width="fill_parent"
   android:layout_height="wrap_content" android:text="Begin Recording"/>

   <Button android:id="@+id/stpBtn" android:layout_width="fill_parent"
   android:layout_height="wrap_content" android:text="Stop Recording"/>

   <Button android:id=
```

```
   "@+id/playRecordingBtn" android:layout_width="fill_parent"
   android:layout_height="wrap_content" android:text="Play Recording"/>

   <Button android:id=
   "@+id/stpPlayingRecordingBtn" android:layout_width="fill_parent"
   android:layout_height="wrap_content" android:text=
      "Stop Playing Recording"/>

   </LinearLayout>

// RecorderActivity.java
import java.io.File;
import android.app.Activity;
import android.media.MediaPlayer;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class RecorderActivity extends Activity {
   private MediaPlayer mediaPlayer;
   private MediaRecorder recorder;
   private static final String OUTPUT_FILE= "/sdcard/recordoutput.3gpp";

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.record);

      Button startBtn = (Button) findViewById(R.id.bgnBtn);

      Button endBtn = (Button) findViewById(R.id.stpBtn);

      Button playRecordingBtn =
            (Button) findViewById(R.id.playRecordingBtn);

      Button stpPlayingRecordingBtn =
            (Button) findViewById(R.id.stpPlayingRecordingBtn);

      startBtn.setOnClickListener(new OnClickListener() {

         @Override
         public void onClick(View view) {
            try {
               beginRecording();
            } catch (Exception e) {
               e.printStackTrace();
            }
```

```
        }
    });

    endBtn.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View view) {
            try {
                stopRecording();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });

    playRecordingBtn.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View view) {
            try {
                playRecording();
            } catch (Exception e) {
                    e.printStackTrace();
            }
        }
    });

    stpPlayingRecordingBtn.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View view) {
            try {
                stopPlayingRecording();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

private void beginRecording() throws Exception {
    killMediaRecorder();

    File outFile = new File(OUTPUT_FILE);

    if(outFile.exists())
    {
        outFile.delete();
    }
```

```
        recorder = new MediaRecorder();
        recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
        recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
        recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
        recorder.setOutputFile(OUTPUT_FILE);
        recorder.prepare();
        recorder.start();

    }
    private void stopRecording() throws Exception {
        if (recorder != null) {
            recorder.stop();
        }
    }

    private void killMediaRecorder() {
        if (recorder != null) {
            recorder.release();
        }
    }

    private void killMediaPlayer() {
        if (mediaPlayer != null) {
            try {
                mediaPlayer.release();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    private void playRecording() throws Exception {
        killMediaPlayer();

        mediaPlayer = new MediaPlayer();
        mediaPlayer.setDataSource(OUTPUT_FILE);

        mediaPlayer.prepare();
        mediaPlayer.start();
    }
    private void stopPlayingRecording() throws Exception {
        if(mediaPlayer!=null)
        {
            mediaPlayer.stop();
        }
    }

    @Override
    protected void onDestroy() {
```

```
        super.onDestroy();

        killMediaRecorder();
        killMediaPlayer();
    }

}
```

Прежде чем углубиться в изучение листинга 9.5, обратите внимание на то, что для записи аудио в файл описания необходимо добавить следующее право доступа:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Выше, в подразделе о картах памяти, мы говорили о том, что если в вашей программе используется minSdkVersion 4 или выше, то также потребуется добавить тег uses-permission для "android.permission.WRITE_EXTERNAL_STORAGE". Наконец, если вы собираетесь испробовать этот пример в эмуляторе, к рабочей станции нужно подключить микрофон.

Внимательно рассмотрев метод onCreate() в листинге 9.5, мы видим, что обработчики щелчков (on-click event handlers) отслеживают сигналы от четырех кнопок. Метод beginRecording() работает с записью. Для записи аудио необходимо создать экземпляр MediaRecorder и задать для него источник аудио, формат вывода, кодировщик аудио и файл вывода. До версии Android SDK 1.6 поддерживался только один источник аудио — микрофон. В Android SDK 1.6 и выше добавилось еще три источника звука, применяемых при мобильной связи. Можно записать целый вызов (MediaRecorder.AudioSource.VOICE_CALL), только восходящий трафик (MediaRecorder.AudioSource.VOICE_UPLINK) или только нисходящий трафик (MediaRecorder.AudioSource.VOICE_DOWNLINK). Восходящим трафиком в данном случае является голос вызывающего абонента, а нисходящим — голос принимающей стороны. Единственный поддерживаемый формат аудио — это 3GPP (3rd Generation Partnership Project). Для кодировщика нужно установить значение AMR_NB, соответствующее узкополосному аудиокодеку AMR (Adaptive Multi-Rate), так как это единственный поддерживаемый вариант. Аудио, записанное в нашем примере, находится на карте памяти в файле /sdcard/recordoutput.3gpp. Обратите внимание: в листинге 9.5 предполагается, что вы создали образ карты памяти и указали на карту памяти эмулятору. Если вы этого не сделали, перечитайте подраздел «Карты памяти», где подробно описано, как это сделать.

В MediaRecorder есть еще несколько методов, которые могут вам пригодиться. Чтобы ограничить длину и размер аудиозаписи, можно использовать методы setMaxDuration(intlength_in_ms) и setMaxFileSize(long length_in_bytes). Максимальная длительность записи задается в миллисекундах, а максимальный размер файла — в байтах. При достижении указанного предела запись останавливается. Оба этих метода появились в Android 1.5, то есть они доступны и на некоторых старых моделях телефонов.

Обратите внимание — в используемых в настоящее время API не поддерживаются потоки. Это значит, что в процессе записи аудио вы не можете получить доступ

к аудиопотоку (например, в целях анализа). Вам придется сначала записать аудиоконтент в файл, а потом работать с ним. В новых версиях Android SDK потоки, скорее всего, будут поддерживаться. Для работы с потоками (streaming) можно попробовать следующий обходной маневр: записать аудио в файл и в ходе записи считывать его из файла в другом потоке (thread) или другой программой.

Изучение видеозаписи

В версии Android SDK 1.5 и выше можно записывать видео при помощи медиафреймворка. Процесс напоминает запись аудио, и на самом деле вместе с видео обычно записывается звуковая дорожка. Но с записью видео связана одна серьезная оговорка. В Android SDK 1.6 и выше для записи видео необходимо предварительно просмотреть кадры с камеры в объекте Surface. В простых программах это не составляет проблемы, так как пользователь обычно и так просматривает информацию, снимаемую камерой. Проблемы начинаются в более сложных программах. Если программа не должна показывать пользователю поток видео «в реальном времени», вам все равно потребуется применять объект Surface, чтобы объект camera мог предварительно просматривать снимаемое видео. Надеемся, что это требование будет смягчено в новых версиях Android SDK, чтобы программа могла записывать информацию прямо в видеобуфер, не копируя ее в компонент пользовательского интерфейса. Но пока у нас нет другого выхода, кроме как работать с Surface, поэтому посмотрим, как это делается (листинг 9.6).

Листинг 9.6. Использование класса MediaRecorder для видеосъемки

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <Button android:id="@+id/bgnBtn" android:layout_width="fill_parent"
      android:layout_height="wrap_content" android:text="Begin Recording"
      android:enabled="false" />

   <Button android:id="@+id/stpBtn" android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Stop Recording" />

   <Button android:id="@+id/playRecordingBtn"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Play Recording" />

   <Button android:id="@+id/stpPlayingRecordingBtn"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
```

```
      android:text="Stop Playing Recording" />

   <RelativeLayout android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:gravity="center">

      <VideoView android:id="@+id/videoView" android:layout_width="176px"
         android:layout_height="144px" />

   </RelativeLayout>
</LinearLayout>

import java.io.File;
import android.app.Activity;
import android.media.MediaRecorder;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends Activity implements
            SurfaceHolder.Callback {

   private MediaRecorder recorder = null;
   private static final String OUTPUT_FILE = "/sdcard/videooutput.mp4";
   private static final String TAG = "RecordVideo";
   private VideoView videoView = null;
   private Button startBtn = null;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      startBtn = (Button) findViewById(R.id.bgnBtn);

      Button endBtn = (Button) findViewById(R.id.stpBtn);

      Button playRecordingBtn = (Button)
            findViewById(R.id.playRecordingBtn);

      Button stpPlayingRecordingBtn =
            (Button) findViewById(R.id.stpPlayingRecordingBtn);

      videoView = (VideoView)this.findViewById(R.id.videoView);

      final SurfaceHolder holder = videoView.getHolder();
```

```
holder.addCallback(this);
holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);

startBtn.setOnClickListener(new OnClickListener() {

   @Override
   public void onClick(View view) {
      try {
         beginRecording(holder);
      } catch (Exception e) {
         Log.e(TAG, e.toString());
         e.printStackTrace();
      }
   }
});

endBtn.setOnClickListener(new OnClickListener() {

   @Override
   public void onClick(View view) {
      try {
         stopRecording();
      } catch (Exception e) {
         Log.e(TAG, e.toString());
         e.printStackTrace();
      }
   }
});

playRecordingBtn.setOnClickListener(new OnClickListener() {

   @Override
   public void onClick(View view) {
      try {
         playRecording();
      } catch (Exception e) {
         Log.e(TAG, e.toString());
         e.printStackTrace();
      }
   }
});

stpPlayingRecordingBtn.setOnClickListener(new OnClickListener() {

   @Override
   public void onClick(View view) {
      try {
         stopPlayingRecording();
      } catch (Exception e) {
         Log.e(TAG, e.toString());
```

```
            e.printStackTrace();
         }
      }
   });
}

@Override
public void surfaceCreated(SurfaceHolder holder) {
   startBtn.setEnabled(true);
}

@Override
public void surfaceDestroyed(SurfaceHolder holder) {
}

@Override
public void surfaceChanged(SurfaceHolder holder, int format, int width,
   int height) {
   Log.v(TAG, „Width x Height = „ + width + „x" + height);
}

private void playRecording() {
   MediaController mc = new MediaController(this);
   videoView.setMediaController(mc);
   videoView.setVideoPath(OUTPUT_FILE);
   videoView.start();
}

private void stopPlayingRecording() {
   videoView.stopPlayback();
}

private void stopRecording() throws Exception {
   if (recorder != null) {
      recorder.stop();
   }
}

@Override
protected void onDestroy() {
   super.onDestroy();
   if (recorder != null) {
      recorder.release();
   }
}

private void beginRecording(SurfaceHolder holder) throws Exception {
   if(recorder!=null)
   {
      recorder.stop();
```

```
            recorder.release();
        }

        File outFile = new File(OUTPUT_FILE);
        if(outFile.exists())
        {
            outFile.delete();
        }

        try {
            recorder = new MediaRecorder();
            recorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);
            recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
            recorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
            recorder.setVideoSize(176, 144);
            recorder.setVideoFrameRate(15);
            recorder.setVideoEncoder(MediaRecorder.VideoEncoder.MPEG_4_SP);
            recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
            recorder.setMaxDuration(30000); // limit to 30 seconds
            recorder.setPreviewDisplay(holder.getSurface());
            recorder.setOutputFile(OUTPUT_FILE);
            recorder.prepare();
            recorder.start();
        }
        catch(Exception e) {
            Log.e(TAG, e.toString());
            e.printStackTrace();
        }
    }
}
```

Как и при записи аудио, которой мы занимались выше, необходимо задать одинаковые права доступа для аудио (android.permission.RECORD_AUDIO) и для карты памяти (android.permission.WRITE_EXTERNAL_STORAGE), но в данном случае мы еще должны указать права доступа для камеры (android.permission.CAMERA). В листинге 9.6 показан класс явления, предоставляющий метод beginRecording() для записи видеоконтента с камеры устройства на карту памяти. Вспомните приведенный выше пример с аудио — в том случае было нужно сначала установить свойства записывающего устройства, а потом вызвать prepare(). Как видите, мы указали в качестве источника видео для MediaRecorder камеру, установленную в устройстве, в качестве источника аудио — микрофон, формат вывода определили как MPEG_4, и т. д. Кроме того, мы задали кодировщики аудио и видео и указали путь к файлу вывода, расположенному на карте памяти, до того, как вызвали методы prepare() и start(). Еще одна вещь, которую надо сделать перед вызовом prepare(), — установить режим предварительного просмотра (preview display) для MediaRecorder. Здесь в игру вступает Surface. В нашем шаблоне есть VideoView, и в режиме предварительного просмотра он может выполнять функцию Surface. Благодаря этому пользователь видит, что именно записывается камерой. Мы устанавливаем заполнитель из VideoView, а потом используем его здесь. Обратите внимание и на то, что

мы активируем кнопку **Begin Recording** (Начать запись) только после создания Surface.

В листинге 9.6 показано взятие видеоинформации с камеры и сохранение ее на карте памяти в файле videooutput.mp4. В настоящее время мы не можем манипулировать контентом, полученным с камеры, — сначала информацию нужно закодировать и сохранить. Однако функция непосредственной манипуляции может появиться в новых версиях Android. Еще одна характерная особенность программы заключается в том, что мы используем один и тот же объект VideoView и для записи видео, и для его воспроизведения. Если вы уже записали видео на карту памяти при помощи этой программы, то сразу после записи можно будет воспроизвести любой видеофрагмент, который уже сохранен на карте памяти как отдельный файл, а ваша программа приспособлена для воспроизведения видеофайлов, расположенных на карте памяти.

Изучение класса MediaStore

До сих пор при работе с медиа мы напрямую инстанцировали классы для воспроизведения и записи медиа в нашем приложении. Одна из великолепных черт Android заключается в том, что вы можете обращаться к другим программам, чтобы они выполняли те или иные задачи. Класс MediaStore предоставляет интерфейс для работы с медиафайлами, сохраненными в устройстве (как на внутреннем ЗУ, так и на внешних носителях). В MediaStore также есть API для обработки медиа. К ним относятся в том числе механизмы, обеспечивающие поиск определенных типов медиа в устройстве, намерения для записи аудио и видео в хранилище данных, методы для составления плей-листов и многое другое. Следует отметить, что этот класс входил и в более ранние версии SDK, но сейчас значительно усовершенствован по сравнению с версией 1.5.

Поскольку класс MediaStore поддерживает намерения, позволяющие записывать аудио и видео, а класс MediaRecorder может делать то же самое, возникает очевидный вопрос: когда использовать MediaStore, а когда — MediaRecorder? В предыдущих примерах, где были продемонстрированы принципы записи аудио и видео, было показано, что MediaRecorder позволяет задавать различные параметры для источника записи. К этим параметрам относятся источник входящего аудио/видео, частота кадров видео, размер кадров видео, форматы вывода и т. д. В MediaStore не предусмотрен такой дифференцированный подход, но вы можете обойтись и без MediaRecorder, если воспользуетесь намерениями MediaStore. Гораздо важнее то, что контент, создаваемый при помощи MediaRecorder, недоступен для других приложений, которые ищут хранилище медиа. При использовании MediaRecorder вы, возможно, захотите добавить запись в хранилище медиа при помощи API, которые предоставляются в MediaStore, и в этом случае лучше сразу работать с MediaStore.

Еще одно важное отличие между двумя классами заключается в том, что при вызове MediaStore через намерение ваша программа не должна запрашивать права доступа на запись аудио, доступ к объекту camera или на сохранение данных на карту памяти. Программа активирует отдельное явление, и вот у этого явления уже должны быть права на запись аудио, доступ к объекту camera или на сохранение

данных на карту памяти. У явлений, относящихся к классу MediaStore, эти права доступа уже есть. Следовательно, ваша программа не обязательно должна их иметь. Теперь посмотрим, как использовать API MediaStore с максимальной эффективностью.

Вы уже знаете, что записывать аудио совсем не сложно. Но ситуация еще упрощается, если применить намерение из MediaStore. В листинге 9.7 показано, как записывать аудио при помощи намерения.

Листинг 9.7. Использование намерения при записи аудио

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
   <Button android:id="@+id/recordBtn"
      android:text="Record Audio"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
</LinearLayout>

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class UsingMediaStoreActivity extends Activity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      setContentView(R.layout.main);

      Button btn = (Button)findViewById(R.id.recordBtn);
      btn.setOnClickListener(new OnClickListener(){

         @Override
         public void onClick(View view) {

            startRecording();

         }});
   }

   public void startRecording() {
      Intent intt =
         new Intent("android.provider.MediaStore.RECORD_SOUND");
```

```
        startActivityForResult(intt, 0);
    }

    @Override
    protected void onActivityResult(int requestCode,
            int resultCode, Intent data) {

        switch (requestCode) {
        case 0:
            if (resultCode == RESULT_OK) {
                Uri recordedAudioPath = data.getData();
            }
        }
    }
}
```

В листинге 9.7 создается намерение, запрашивающее у системы право начать запись аудио. Код применяет намерение к явлению, вызывая `startActivityForResult()` и передавая намерение и `requestCode`. Когда работа запрошенного явления завершится, вызывается `onActivityResult()` с `requestCode`. В `onActivityResult()` показано, что мы ищем `requestCode`, совпадающий с кодом, переданным `startActivityForResult()`, а затем находим URI сохраненного медиафайла, вызывая `data.getData()`. После этого URI можно подать в намерение и прослушать запись. Пользовательский интерфейс, соответствующий листингу 9.7, показан на рис. 9.6.



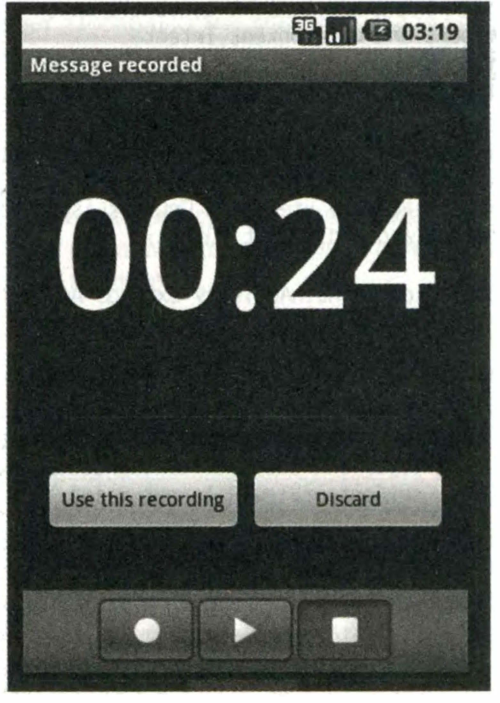


Рис. 9.6. Встроенное приложение для аудиозаписи — до *(слева)* и после записи *(справа)*

На рис. 9.6 показаны два скриншота. На изображении *слева* мы видим программу в процессе записи, а *справа* показано, как выглядит пользовательский интерфейс явления, когда запись уже завершена.

Подобно тому как в предыдущем примере мы использовали намерение для записи аудио, теперь мы применим намерение для съемки изображения. В листинге 9.8 показано, как это сделать.

Листинг 9.8. Запуск намерения для съемки изображения

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >

   <Button android:id="@+id/btn"
      android:text="Take Picture"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="captureImage" />

</LinearLayout>

import android.app.Activity;
import android.content.ContentValues;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.provider.MediaStore.Images.Media;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {

   Uri myPicture = null;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
   }

   public void captureImage(View view)
   {
      ContentValues values = new ContentValues();
      values.put(Media.TITLE, "My demo image");
```

```
        values.put(Media.DESCRIPTION,
              "Image Captured by Camera via an Intent");

        myPicture = getContentResolver().insert
              (Media.EXTERNAL_CONTENT_URI, values);

        Intent i = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        i.putExtra(MediaStore.EXTRA_OUTPUT, myPicture);

        startActivityForResult(i, 0);
    }

    @Override
    protected void onActivityResult(int requestCode,
              int resultCode, Intent data) {
        if(requestCode==0 && resultCode==Activity.RESULT_OK)
        {
            // Теперь мы знаем, что URI myPicture указывает
            // на только что снятое изображение.
        }
    }
}
```

В классе явления, показанном в листинге 9.8, определяется метод captureImage(). В нем создается намерение, название действия которого — MediaStore.ACTION_IMAGE_CAPTURE. При запуске этого намерения приложение camera переводится в приоритетный режим и пользователь фотографирует. Поскольку мы заранее создали URI, мы можем сообщить дополнительную информацию об изображении до того, как камера его снимет. Дополнительную информацию добавляет класс ContentValues. К values можно добавлять не только TITLE и DESCRIPTION, но и другие атрибуты. Посмотрите описание MediaStore.Images.ImageColumns в справке по Android, там дается полный список атрибутов. После того как изображение сфотографировано, делается обратный вызов onActivityResult(). В данном примере мы использовали поставщик медиаконтента, чтобы создать новый файл. Мы также могли создать URI нового файла, находящегося на карте памяти, вот так:

```
myPicture = Uri.fromFile(new File("/sdcard/DCIM/100ANDRO/imageCaptureIntent.jpg"));
```

Правда, при создании URI таким способом будет непросто установить для изображения атрибуты, например TITLE и DESCRIPTION. Существует другой способ активации намерения камеры, чтобы снять изображение. Если мы не пошлем с намерением никакого URI, в аргументе намерения для onActivityResult() будет возвращен растровый (bitmap) объект. При применении такого метода возникает проблема: по умолчанию растровый рисунок в таком случае обычно гораздо меньше оригинала, потому что разработчики Android не позволяют вам передавать большие объемы данных из явления камеры в явление вашей программы. Размер растрового рисунка будет равен 50 Кбайт. Чтобы получить объект Bitmap, в методе onActivityResult() нужно осуществить примерно такую операцию:

```
Bitmap myBitmap = (Bitmap) data.getExtras().get("data");
```

В MediaStore есть также намерение для съемки видео, которое работает сходным образом. Для видеосъемки можно применять MediaStore.ACTION_VIDEO_CAPTURE.

Добавление медийного контента в MediaStore

Еще одна замечательная функция медийного фреймворка Android — это возможность добавлять информацию о контенте в хранилище медиа. Для этого применяется класс MediaScannerConnection. Иными словами, если хранилище медиа «не знает», что в нем появилась новая информация, то мы используем MediaScannerConnection, чтобы сообщить хранилищу о том, что она есть. После этого данный образец контента может обрабатываться так же, как и все остальные. Рассмотрим, как это работает (листинг 9.9).

Листинг 9.9. Добавление файла в MediaStore

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content">

   <EditText android:id="@+id/fileName"
      android:hint="Enter new filename"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />

   <Button android:id="@+id/scanBtn"
      android:text="Add file"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="startScan" />

</LinearLayout>

import java.io.File;
import android.app.Activity;
import android.content.Intent;
import android.media.MediaScannerConnection;
import android.media.MediaScannerConnection.MediaScannerConnectionClient;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MediaScannerActivity extends Activity implements
MediaScannerConnectionClient
{
   private EditText editText = null;
   private String filename = null;
```

```
    private MediaScannerConnection conn;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        editText = (EditText)findViewById(R.id.fileName);
    }

    public void startScan(View view)
    {
        if(conn!=null)
        {
            conn.disconnect();
        }

        filename = editText.getText().toString();

        File fileCheck = new File(filename);
        if(fileCheck.isFile()) {
            conn = new MediaScannerConnection(this, this);
            conn.connect();
        }
        else {
            Toast.makeText(this,
            "That file does not exist",
            Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onMediaScannerConnected() {
        conn.scanFile(filename, null);
    }

    @Override
    public void onScanCompleted(String path, Uri uri) {
        try {
            if (uri != null) {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(uri);
                startActivity(intent);
            }
            else {
                Log.e("MediaScannerDemo", "That file is no good");
            }
        } finally {
            conn.disconnect();
            conn = null;
        }
    }
}
```

В листинге 9.9 показан класс явления, добавляющий файл в MediaStore. Если файл добавлен успешно, он демонстрируется пользователю; для этого применяется намерение. В это время «за кулисами» файл проверяется MediaScanner, который определяет тип данного файла и важную информацию о нем. Конечно, мы можем указать MediaScanner тип MIME в качестве второго аргумента scanFile(). Если MediaScanner не удается определить тип файла по его расширению, файл не добавляется в хранилище. Если файл относится к MediaStore, в базе данных поставщика медиа создается новая запись. Сам файл никуда не перемещается. Но теперь поставщику медиа известно о существовании этого файла. Если вы добавили файл-изображение, его теперь можно открыть и просмотреть в программе Gallery. Если добавленный файл — музыкальный, его можно увидеть в программе Music.

Если вы захотите просмотреть базу данных поставщика медиа, откройте окно инструментов, запустите adb shell и перейдите в /data/data/com.android.providers.media/databases. Здесь вы найдете базы данных, среди которых будет internal.db. Там же могут находиться файлы внешних баз данных, соответствующие одной или нескольким картам памяти. Поскольку к телефону Android можно подключать несколько карт памяти, в этом каталоге может находиться несколько файлов внешних баз данных. Утилита sqlite3 используется для просмотра таблиц этих баз данных. Это таблицы для аудио, изображений и видео. В главе 3 работа с sqlite3 рассмотрена более подробно.

На этом мы завершаем беседу о медийных API. Надеемся, нам удалось убедить вас в том, что записывать и воспроизводить медиаконтент в Android совсем не сложно. Теперь займемся API, которые обеспечивают выполнение функций телефонии.

Использование API, обеспечивающих выполнение функций телефонии

В этом разделе займемся изучением функций телефонии в Android и соответствующих интерфейсов API. В частности, мы расскажем, как посылать и получать SMS, а также как устанавливать и завершать телефонные вызовы. Начнем с SMS.

Работа с SMS

SMS (Short Message Service, CMC) означает «служба коротких сообщений». Эту службу также часто называют *обменом текстовыми сообщениями*. В Android SDK поддерживаются функции отправки и получения текстовых сообщений. Сначала обсудим, какие способы отправки текстовых сообщений предусмотрены в Android SDK.

Отправка SMS

Чтобы посылать из программы текстовые сообщения, нужно добавить в файл описания право доступа <uses-permission android:name="android.permission.SEND_SMS" />, а затем использовать класс android.telephony.SmsManager (листинг 9.10).

Листинг 9.10. Отправка коротких (текстовых) сообщений

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <TextView android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Destination Address:" />

      <EditText android:id="@+id/addrEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:phoneNumber="true" android:text="9045551212" />

   </LinearLayout>

   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="wrap_content">

      <TextView android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Text Message:" />

      <EditText android:id="@+id/msgEditText"
         android:layout_width="fill_parent"
         android:layout_height="wrap_content"
         android:text="hello sms" />

   </LinearLayout>

   <Button android:id="@+id/sendSmsBtn" android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Send Text Message" />
</LinearLayout>

import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
```

```
import android.widget.Toast;

public class TelephonyDemo extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

   setContentView(R.layout.main);

   Button sendBtn = (Button)findViewById(R.id.sendSmsBtn);
   sendBtn.setOnClickListener(new OnClickListener(){

      @Override
      public void onClick(View view) {
         EditText addrTxt =
            (EditText)TelephonyDemo.this.findViewById
                  (R.id.addrEditText);

         EditText msgTxt =
            (EditText)TelephonyDemo.this.findViewById(R.id.msgEditText);

         try {
            sendSmsMessage(addrTxt.getText().toString(),
                  msgTxt.getText().toString());
            Toast.makeText(TelephonyDemo.this, "SMS Sent",
            Toast.LENGTH_LONG).show();
         } catch (Exception e) {
            Toast.makeText(TelephonyDemo.this, "Failed to send SMS",
                  Toast.LENGTH_LONG).show();
            e.printStackTrace();
         }
      }});
   }

   @Override
   protected void onDestroy() {
      super.onDestroy();
   }

   private void sendSmsMessage(String address,
            String message)throws Exception
   {
      SmsManager smsMgr = SmsManager.getDefault();
      smsMgr.sendTextMessage(address, null, message, null, null);
   }
}
```

В листинге 9.10 показано, как в Android SDK выполняется отправка SMS. Изучив фрагмент, в котором представлен шаблон, можно увидеть, что в пользова-

тельском интерфейсе есть два поля EditText: в одном указывается адрес получателя SMS (то есть его телефонный номер), а в другом содержится текстовое сообщение. В пользовательском интерфейсе также есть кнопка для отправки SMS. Все это показано на рис. 9.7.

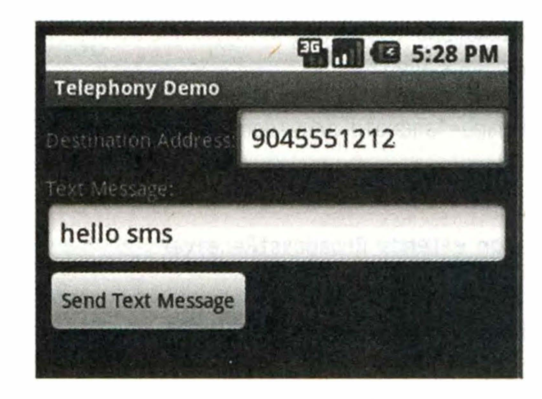


Рис. 9.7. Пользовательский интерфейс для отправки SMS

В этом примере есть очень интересный метод sendSmsMessage(). В нем используется метод sendTextMessage() класса SmsManager, предназначенный для отправки SMS. Ниже показан образец SmsManager.sendTextMessage():

```
sendTextMessage(String destinationAddress, String smscAddress,
String textMsg, PendingIntent sentIntent, PendingIntent deliveryIntent);
```

В данном примере указывается только номер получателя и параметры текстового сообщения. Однако этот метод можно перенастроить так, чтобы он не использовал заданный по умолчанию центр SMS (адрес сервера мобильного оператора, с которого SMS направляются адресатам). Кроме того, можно задать и такие настройки, благодаря которым при отправке сообщения вызываются «ждущие» намерения (pending intents) и в программу приходит уведомление о доставке сообщения.

Посылать SMS не сложнее, чем пользоваться другими функциями Android. Однако не забывайте, что при работе с эмулятором ваши SMS не уходят к адресату. Если метод sendTextMessage() выполняется без исключения, то в реальных условиях отправка SMS должна пройти успешно. В листинге 9.10 показано, что для отображения SMS в пользовательском интерфейсе используется класс Toast — с его помощью также можно узнать, успешно ли прошла отправка сообщения.

Отправка SMS — это только полдела. Теперь нужно научиться отслеживать входящие SMS.

Отслеживание входящих SMS-сообщений

Работа с входящими SMS начинается с запроса права доступа, позволяющего получать SMS. Воспользуемся только что написанной программой, при помощи которой мы отправляли SMS. Добавим в файл описания право доступа <uses-permission android:name="android.permission.RECEIVE_SMS" />. Далее потребуется

внедрить контролирующую программу (monitor), которая будет ожидать SMS. Для этого задайте BroadcastReceiver для действия <action android:value="android.provider.Telephony.SMS_RECEIVED" />. Чтобы внедрить приемник, напишем класс, дополняющий android.content.BroadcastReceiver, а потом зарегистрируем его в файле описания (внутри тега <application>). Эта операция показана в листинге 9.11.

Листинг 9.11. Отслеживание SMS-сообщений

```
<receiver android:name="MySMSMonitor">
   <intent-filter>
      <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
   </intent-filter>
</receiver>

public class MySMSMonitor extends BroadcastReceiver
{
   private static final String ACTION =
         "android.provider.Telephony.SMS_RECEIVED";
   @Override
   public void onReceive(Context context, Intent intent)
   {
      if(intent!=null && intent.getAction()!=null &&
            ACTION.compareToIgnoreCase(intent.getAction())==0)
      {
         Object[]pduArray= (Object[]) intent.getExtras().get("pdus");
         SmsMessage[] messages = new SmsMessage[pduArray.length];
         for (int i = 0; i<pduArray.length; i++) {
            messages[i] = SmsMessage.createFromPdu (
                     (byte[])pduArray [i]);
         }
         Log.d("MySMSMonitor","SMS Message Received.");
      }
   }
}
```

Верхняя часть листинга 9.11 — это описание BroadcastReceiver, предназначенного для получения SMS. Класс программы, контролирующей прием SMS, называется MySMSMonitor. В нем реализуется абстрактный метод onReceive(), вызываемый системой всякий раз при приходе SMS. Получившееся приложение можно протестировать, перейдя в программе Eclipse в режим **Emulator Control** (Управление эмулятором). Запустите программу в эмуляторе, а потом перейдите в **Window** ▸ **Show View** ▸ **Other** ▸ **Android** ▸ **Emulator Control** (Окно ▸ Режим просмотра ▸ Другие ▸ Android ▸ Управление эмулятором). Пользовательский интерфейс позволяет посылать данные на эмулятор, имитируя, таким образом, прием SMS или телефонного вызова. На рис. 9.8 показано, как послать SMS на эмулятор. Для этого нужно заполнить поле **Incoming number** (Входящий номер), а затем установить переключатель в положение **SMS**. Потом необходимо написать какой-нибудь текст в поле **Message** (Сообщение) и нажать кнопку **Send** (Отправить). После этого SMS будет отправлена в эмулятор. При этом активируется метод BroadcastReceiver's onReceive().

Рис. 9.8. Применение пользовательского интерфейса «Управление эмулятором» для отправки на эмулятор SMS-сообщений

В методе `onReceive()` будет широковещательное намерение (broadcast intent), в свойстве `bundle` которого будет находиться `SmsMessage`. Можно извлечь `SmsMessage`, вызвав `intent.getExtras().get("pdus")`. Такой вызов возвращает массив объектов, описанных в формате PDU (Protocol Description Unit, единица описания протокола). Это промышленный стандарт представления SMS-сообщений. Затем можно преобразовывать сообщения в формате PDU в объекты `SmsMessage`, по размеру равные массиву PDU. Наконец, проводится итерация массива PDU, в ходе которой из массива при помощи метода `SmsMessage.createFromPdu()` создаются объекты `SmsMessage`. После прочтения входящего сообщения остается сделать совсем немного. Широковещательный приемник получает в системе наивысший приоритет, но должен выполнить задачу быстро и не выводится на передний план, то есть пользователь не замечает его работы. Следовательно, вы располагаете ограниченным количеством вариантов. Не нужно никакой непосредственной работы с пользовательским интерфейсом. Достаточно выдать уведомление (notification), так как оно активирует службу, которая в фоновом режиме продолжит выполнение задачи. После того как метод `onReceive()` завершит работу, несущий процесс этого метода можно завершить в любой момент. Запуск службы оправдан, но не нужно делать связывания с ней, так как в последнем случае ваш процесс, возможно, должен будет просуществовать некоторое время (пока работает служба), а этого времени может не быть.

Теперь вернемся к обсуждению SMS и посмотрим, как строится работа с различными каталогами SMS.

Работа с каталогами SMS

Как правило, пользователю нужен доступ к каталогу входящих сообщений. Для этого сначала нужно добавить в файл описания право доступа, обеспечивающее чтение сообщений (`<uses-permission android:name="android.permission.READ_SMS"/>`). Это право доступа позволяет пользователю читать входящие сообщения.

Чтобы прочесть сообщение, нужно выполнить запрос к каталогу входящих сообщений. В листинге 9.12 показано, как это делается.

Листинг 9.12. Отображение SMS из каталога входящих сообщений

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/sms_inbox.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
   <TextView android:id="@+id/row"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"/>

</LinearLayout>

import android.app.ListActivity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.widget.ListAdapter;
import android.widget.SimpleCursorAdapter;

public class SMSInboxDemo extends ListActivity {

   private ListAdapter adapter;
   private static final Uri SMS_INBOX = Uri.parse("content://sms/inbox");

   @Override
   public void onCreate(Bundle bundle) {
      super.onCreate(bundle);
      Cursor c = getContentResolver()
            .query(SMS_INBOX, null, null, null, null);
      startManagingCursor(c);
      String[] columns = new String[] { "body" };
      int[] names = new int[] { R.id.row };
      adapter = new SimpleCursorAdapter(this, R.layout.sms_inbox,
            c, columns, names);

      setListAdapter(adapter);
   }
}
```

В листинге 9.12 мы открываем каталог входящих сообщений и создаем список, в каждом элементе которого содержится тело SMS. В той части листинга 9.12, где представлен шаблон, находится простой TextView, заключающий тело каждого сообщения в элемент списка. Чтобы получить список SMS, создается URI, указывающий на каталог входящих SMS (content://sms/inbox), а потом выполняется простой запрос. После этого тело SMS фильтруется и устанавливается адаптер

списка для ListActivity. Когда код из листинга 9.12 будет выполнен, в каталоге входящих появится список SMS. Прежде чем запускать код в эмуляторе, убедитесь, что SMS создавались при помощи Emulator Control.

Теперь, имея доступ к каталогу входящих сообщений, нужно открыть доступ и к другим каталогам, в частности к отправленным сообщениям и черновикам. Разница между процедурами доступа к каталогу входящих сообщений и к другим каталогам заключается только в указываемом URI. Например, чтобы попасть в каталог отправленных сообщений выполните запрос к content://sms/sent. Далее приведен полный список каталогов коротких сообщений и соответствующих URI.

- *Все* — content://sms/all.
- *Входящие* — content://sms/inbox.
- *Отправленные* — content://sms/sent.
- *Черновики* — content://sms/draft.
- *Исходящие* — content://sms/outbox.
- *Недошедшие* — content://sms/failed.
- *Ожидающие* — content://sms/queued.
- *Недоставленные* — content://sms/undelivered.
- *Чат* — content://sms/conversations.

В Android совместно обрабатываются сообщения SMS и MMS, вы получаете доступ к поставщикам содержимого сообщений обоих видов одновременно, указывая AUTHORITY mms-sms. Следовательно, доступ к URI будет выглядеть так:

```
content://mms-sms/conversations
```

Отправка электронной почты

Итак, мы изучили, как отправлять SMS. Логично предположить, что в системе есть похожий API, предназначенный для отправки электронной почты. Но, к сожалению, в Android такого API нет. Основная причина этого — пользователь вряд ли захочет иметь программу, которая будет самопроизвольно отправлять электронные письма. Чтобы отправлять электронную почту, пользуйтесь зарегистрированными программами. Такую программу можно запустить, например, при помощи ACTION_SEND:

```
Intent emailIntent=new Intent(Intent.ACTION_SEND);

String subject = "Hi!";
String body = "hello from android....";

String[] extra = new String[]{"aaa@bbb.com"};
emailIntent.putExtra(Intent.EXTRA_EMAIL, extra);

emailIntent.putExtra(Intent.EXTRA_SUBJECT, subject);
emailIntent.putExtra(Intent.EXTRA_TEXT, body);
emailIntent.setType("message/rfc822");

startActivity(emailIntent);
```

Код запускает стандартную почтовую программу, и пользователь сам решает, посылать письмо или нет. К намерению электронной почты можно добавить дополнительные параметры EXTRA_CC и EXTRA_BCC.

Теперь поговорим о диспетчере телефонии.

Работа с диспетчером телефонии

API для работы с функциями телефонии включает в себя специальный диспетчер (android.telephony.TelephonyManager), при помощи которого можно получать информацию о телефонных службах, работающих на устройстве, получать абонентскую информацию и регистрироваться для изменения состояний телефонии. Обычно для работы телефонной программы требуется, чтобы приложение применяло к входящим вызовам бизнес-логику. Например, при поступлении вызова следует остановить работу плеера и возобновить ее, когда вызов будет завершен. Итак, в этом разделе будет показано, как регистрировать изменения состояний телефонии и как обнаруживать входящие телефонные вызовы. Подробности изложены в листинге 9.13.

Листинг 9.13. Работа с диспетчером телефонии

```
public class TelephonyServiceDemo extends Activity
{
   private static final String TAG="TelephonyServiceDemo";
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);

      TelephonyManager teleMgr =
         (TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
      teleMgr.listen(new MyPhoneStateListener(),
         PhoneStateListener.LISTEN_CALL_STATE);
   }

   class MyPhoneStateListener extends PhoneStateListener
   {
      @Override
      public void onCallStateChanged(int state, String incomingNumber) {
         super.onCallStateChanged(state, incomingNumber);

         switch(state)
         {

            case TelephonyManager.CALL_STATE_IDLE:
               Log.d(TAG, "call state idle...incoming number is["+
                     incomingNumber+"]");break;
            case TelephonyManager.CALL_STATE_RINGING:
               Log.d(TAG, "call state ringing...incoming number is["+
                     incomingNumber+"]");break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
```

```
                Log.d(TAG, "call state Offhook...incoming number is["+
                        incomingNumber+"]");break;
            default:
                Log.d(TAG, "call state ["+state+"]incoming number is["+
                        incomingNumber+"]");break;
        }
    }
  }
}
```

Для работы с диспетчером телефонии необходимо добавить в файл описания право доступа `<uses-permission android:name="android.permission.READ_PHONE_STATE" />`, которое позволяет получать информацию о состоянии телефона. В листинге 9.13 показано, что для получения сообщений о состоянии телефона нужно внедрить `PhoneStateListener` и вызывать метод `listen()`, относящийся к `TelephonyManager`. При получении вызова или изменении состояния телефона система вызывает метод `onCallStateChanged()` из `PhoneStateListener`, в котором передается новое состояние и номер входящего вызова. В случае с входящим вызовом требуется состояние `CALL_STATE_RINGING`. В данном примере в файл регистрации заносится отладочное сообщение, но в конкретной программе вместо этого может срабатывать специальная бизнес-логика. Входящие вызовы можно моделировать в пользовательском интерфейсе управления эмулятором Eclipse, с которым мы уже работали в разделе об SMS (см. рис. 9.8).

При работе с изменениями состояний телефона вам также может быть нужен номер абонента (пользователя). Этот номер возвращается методом `TelephonyManager.getLine1Number()`.

Резюме

В этой главе мы поговорили о медийном фреймворке Android и об API для работы с функциями телефонии. В разделе о медиа было показано, как воспроизводится аудио и видео. Мы также рассмотрели запись аудио и видео — непосредственно и при помощи намерений.

Из второй части главы вы узнали, как в Android реализуются службы телефонии. Мы показали, как отправлять текстовые сообщения и как отслеживать входящие сообщения. Кроме того, мы рассмотрели, как получить доступ к различным каталогам SMS, расположенным в устройстве. В заключение мы поговорили о классе `TelephonyManager`.

В следующей главе мы займемся трехмерной графикой и узнаем, как в приложениях Android используется OpenGL.

10 Программирование трехмерной графики при помощи OpenGL

В этой главе мы подробно поговорим о работе с графическим интерфейсом прикладного программирования (API) OpenGL ES на платформе Android.

OpenGL ES — это специальная версия OpenGL, оптимизированная для работы со встроенными системами и другими маломощными устройствами, в частности мобильными телефонами. На платформе Android поддерживается версия OpenGL ES 1.0. В дистрибутиве Android SDK содержатся образцы OpenGL ES. Однако документация о том, как начать работу с OpenGL ES, в SDK практически отсутствует. Поскольку OpenGL является открытым стандартом, предполагается, что программисты могут научиться работе с ним из источников, не связанных напрямую с Android. Побочным эффектом такого допущения является недостаток онлайн-ресурсов или образцов кода Android, описывающих работу с OpenGL на этой платформе, а те примеры, которые можно найти, уже требуют определенных знаний в области OpenGL.

В данной главе мы постараемся прояснить эти моменты. Если вы имеете хотя бы базовое представление об OpenGL, то к концу этой главы вы сможете неплохо программировать в OpenGL. В данной главе нам удалось обойтись практически без математики (в других книгах по OpenGL ее достаточно много).

В первой части главы будет сделан обзор OpenGL, OpenGL ES и некоторых альтернативных стандартов.

Во второй части мы объясним теоретические основы OpenGL. Если вы ранее не работали с OpenGL, то прочитать этот раздел необходимо. В этом разделе будет рассмотрено, как в OpenGL применяются координаты, идея камеры и рисовальные API (drawing API), очень важные при работе с OpenGL.

В третьей части главы мы расскажем, как в Android строится взаимодействие с API OpenGL ES. В этом разделе будет рассмотрен `GLSurfaceView` и интерфейс `Renderer`, а также их совместное использование при рисовании в OpenGL. Здесь мы покажем несколько элементарных примеров — нарисуем простой треугольник и рассмотрим, как влияет на этот рисунок внесение изменений в API настройки сцены в OpenGL.

ПРИМЕЧАНИЕ

Концепция камеры в OpenGL напоминает класс Camera из графического пакета Android, изученный в главе 6, но в то же время отличается от него. Объект Camera из графического пакета Android имитирует трехмерный просмотр изображений, проецируя движущиеся двухмерные изображения в трехмерное пространство. В свою очередь, камера OpenGL — это феномен, представляющий собой виртуальную точку наблюдения. Иными словами, камера OpenGL моделирует реальную сцену, открывая просмотр объектов с позиции наблюдателя, который смотрит на них через камеру. Подробнее эти вопросы рассматриваются в подразделе «Камера и координаты в OpenGL» раздела «Основы OpenGL». Объект Camera и камера OpenGL никак не связаны с обычной материальной камерой, которая позволяет снимать видео.

В четвертом разделе мы более углубленно изучим OpenGL ES и сделаем вводное описание идеи контуров (shapes). В этом же разделе мы рассмотрим текстуры и покажем, как рисовать несколько фигур при помощи одного метода draw.

В заключение перечислим ресурсы, которыми мы пользовались при подготовке материала для этой главы.

Начнем с истории и основ OpenGL.

История и основы OpenGL

OpenGL (первоначальное название — Open Graphics Library, открытая графическая библиотека) была создана для рабочих станций Unix компанией Silicon Graphics, Inc. (SGI). Первая стандартизированная спецификация OpenGL появилась только в 1992 году, хотя версия этой библиотеки авторства SGI использовалась уже довольно долго. В наши дни стандарт OpenGL широко применяется во всех операционных системах и служит основой для большинства игр, а также таких отраслей, как системы автоматизированного проектирования (САПР) и даже виртуальная реальность (virtual reality, VR).

В настоящее время стандартом OpenGL управляет промышленный консорциум, называемый Khronos Group (http://www.khronos.org), основанный в 2000 году. У его истоков стояли такие компании, как NVIDIA, Sun Microsystems, ATI Technologies и SGI. Спецификация OpenGL подробно изложена на сайте консорциума: http://www.khronos.org/opengl.

Официальная документация по OpenGL находится здесь: http://www.opengl.org/documentation. На этой странице вам предоставляется доступ к выложенным в Сети книгам и онлайн-ресурсам, которые посвящены OpenGL. Наиболее классическим трудом по этой теме считается Programming Guide: The Official Guide to Learning OpenGL, Version 1.1, называемая также «Красной книгой» OpenGL. Онлайн-версия этого издания находится по следующему адресу: http://www.glprogramming.com/red.

Данная книга хороша в содержательном отношении и неплохо читается. Правда, нам понадобилось немного попотеть, чтобы разобраться с сущностями units *(единицы)* и coordinates *(координаты)*, которые используются при рисовании. Мы попытаемся понятно изложить эти идеи, так как с их помощью описывается, что вы рисуете и что видите в OpenGL. Основной областью применения этих феноменов является настройка камеры OpenGL, а также настройка *поля видимости*

(viewing box), которое также называется *зоной видимости* (viewing volume) или *видимым пространством* (frustum).

Поскольку предметом нашей главы является OpenGL, мы должны немного поговорить и о Direct3D, компоненте API DirectX, используемого в программах Microsoft. Похоже, что Direct3D становится стандартной технологией на мобильных устройствах, работающих под Windows. Более того, поскольку OpenGL и Direct3D похожи, можно даже читать книги по Direct3D, чтобы лучше понимать, как реализуется трехмерное рисование.

Стандарт Direct3D, введенный Microsoft в 1996 году, программируется с применением интерфейсов COM (Component Object Model, компонентная объектная модель). В мире Windows интерфейсы COM применяются для обмена информацией между различными компонентами приложения. Когда компонент разработан и предоставлен для использования через интерфейс COM, к нему открывается доступ для любого языка разработки платформы Windows, как извне приложения, так и изнутри. В мире Unix аналогом COM является CORBA (Common Object Request Broker Architecture, единая архитектура программы-брокера объектных запросов).

С другой стороны, в OpenGL используются языковые привязки, напоминающие аналогичные элементы языка C. Языковая привязка позволяет применять обычную библиотеку несколькими различными языками, например C, C++, Visual Basic, Java и т. д.

Теперь сосредоточимся на OpenGL ES — версии OpenGL, специально предназначенной для мобильных платформ.

OpenGL ES

Авторству Khronos Group принадлежат еще два стандарта, тесно связанных с OpenGL: OpenGL ES и EGL Native Platform Graphics Interface, обычно называемый просто EGL. Как было указано выше, OpenGL ES — это уменьшенная версия OpenGL, предназначенная для работы со встроенными системами.

ПРИМЕЧАНИЕ

Сообщество Java Community Process также занимается разработкой объектно-ориентированной абстракции OpenGL для мобильных устройств, называемой Mobile 3D Graphics (M3G). Мы кратко опишем M3G в подразделе «M3G — еще один стандарт трехмерной графики, применяемый в Java ME».

В целом интерфейс EGL обеспечивает взаимодействие между базовой операционной системой и отображающими API, входящими в состав OpenGL ES. Поскольку OpenGL и OpenGL ES — это интерфейсы общего назначения, применяемые при рисовании, в каждой операционной системе должна предоставляться стандартная среда выполнения программ (hosting environment), с которой будут взаимодействовать OpenGL и OpenGL ES. В Android SDK версий 1.5 и выше специфика этих платформ реализована довольно неплохо. Мы займемся этим вопросом в разделе, который называется «Взаимодействие OpenGL ES и Android».

Целевыми устройствами для OpenGL ES являются сотовые телефоны, радиоэлектронные устройства и даже транспортные средства. Поскольку OpenGL ES должна быть гораздо меньше OpenGL, многие удобные функции из нее удалены. Например, в OpenGL ES отсутствует непосредственная поддержка рисования

прямоугольников. Здесь сначала рисуются два треугольника, из которых потом составляется прямоугольник.

При изучении поддержки OpenGL в Android мы займемся в первую очередь OpenGL ES и ее привязками к ОС Android, работающими через Java и EGL. Документация по OpenGL ES находится здесь: http://www.khronos.org/opengles/documentation/opengles1_0/html/index.html.

По мере изучения материала главы мы будем возвращаться к этой ссылке, так как в данном документе определяются и описываются все API OpenGL ES, а также аргументы для каждого API. Эти интерфейсы похожи на API, применяемые в Java, поэтому в данной главе мы кратко расскажем о самых важных из них.

OpenGL ES и Java ME

OpenGL ES, как и OpenGL, — это «плоский» API на базе языка С. Поскольку интерфейс прикладного программирования Android SDK основан на Java, при работе с Android нужно выполнить привязку к OpenGL ES. В Java ME такая привязка уже определяется в запросе на спецификацию (JSR) 239: привязка Java к OpenGL ES API. JSR 239, в свою очередь, основан на JSR 231, при помощи которого осуществляется привязка к OpenGL версии 1.5. В узком смысле JSR 239 можно считать подмножеством JSR 231, но на самом деле это не так, поскольку в OpenGL ES есть некоторые расширения, отсутствующие в версии OpenGL 1.5.

Документация по JSR 239 находится по адресу http://java.sun.com/javame/reference/apis/jsr239. Данный документ позволит вам составить представление о том, какие API имеются в OpenGL ES. Кроме того, там содержится ценная информация о следующих пакетах:

- javax.microedition.khronos.egl;
- javax.microedition.khronos.opengles;
- java.nio.

Пакет nio необходим. Это объясняется тем, что ради повышения эффективности экземпляры OpenGL ES принимают в качестве ввода только байтовые потоки. В пакете nio определяется много возможностей подготовки нативных буферов для работы с OpenGL. Некоторые из этих API показаны в действии в пункте «glVertexPointer и указание вершин при рисовании» подраздела «Важнейшие приемы рисования при помощи OpenGL ES».

Документация о поддержке OpenGL в Android SDK (правда, очень скудная) имеется по следующей ссылке: http://developer.android.com/guide/topics/graphics/opengl.html. В этом документе сказано, что поддержка OpenGL в Android в целом аналогична JSR 239, но в некоторых случаях может отличаться от этого запроса на спецификацию.

M3G — еще один стандарт трехмерной графики, применяемый в Java ME

JSR 239 — это просто языковая привязка между Java и нативным стандартом OpenGL ES. Как было кратко упомянуто в подразделе «OpenGL ES», в Java

имеется иной API для работы с 3D-графикой на мобильных устройствах — M3G. Этот объектно-ориентированный стандарт определяется в запросах на спецификацию JSR 184 и JSR 297, второй используется реже. Согласно JSR 184, M3G является облегченным объектно-ориентированным интерактивным API трехмерной графики, предназначенным для мобильных устройств.

Объектно-ориентированная природа M3G отличает его от OpenGL ES. Подробное описание содержится на домашней странице JSR 184: http://www.jcp.org/en/jsr/detail?id=184.

API для M3G имеются в пакете Java.

M3G является более высокоуровневым API, чем OpenGL ES, то есть его проще изучить. Однако пока нет однозначной оценки того, насколько хорошо этот стандарт будет работать в мобильных устройствах. Итак, мы обсудили OpenGL ES и кратко затронули стандарт M3G. Теперь перейдем к изучению фундаментальных основ OpenGL ES.

Основы OpenGL

В этом разделе будет рассказано о концепциях, лежащих в основе OpenGL и OpenGL ES. Мы изучим все основные API. Чтобы дополнить информацию, содержащуюся в этой главе, можете обратиться к подразделу «Ресурсы по OpenGL», расположенному ближе к ее концу. Среди перечисленных там ресурсов — «Красная книга», документация по JSR 239 и справка по Khronos Group API.

ПРИМЕЧАНИЕ

Приступив к работе с ресурсами по OpenGL, вы заметите, что некоторые API отсутствуют в OpenGL ES. Вот здесь вам и пригодится справочный мануал Khronos Group по OpenGL ES.

Следующие API будут рассмотрены достаточно подробно, так как они имеют определяющее значение для понимания OpenGL и OpenGL ES:

- `glVertexPointer`;
- `glDrawElements`;
- `glColor`;
- `glClear`;
- `gluLookAt`;
- `glFrustum`;
- `glViewport`.

Изучив эти API, вы узнаете:

- как использовать важнейшие API OpenGL ES, предназначенные для рисования;
- как очищать палитру;
- как указывать цвета;
- что такое «камера» и «координаты» применительно к OpenGL ES.

Важнейшие приемы рисования при помощи OpenGL ES

В OpenGL ES вы рисуете в трехмерном пространстве. Для начала задается серия точек, которые еще называются «вершинами» (vertex). Каждая из этих точек имеет три значения: координаты *x*, *y* и *z*.

Затем эти точки объединяются в контуры (shapes). Точки можно объединять в так называемые *примитивные контуры* — кроме самих точек, к ним в OpenGL ES относятся линии и треугольники. Обратите внимание — в OpenGL к примитивным контурам также относятся прямоугольники и многоугольники. Продолжая работать с OpenGL и OpenGL ES, вы будете встречать и другие различия, причем во втором варианте всегда будет меньше элементов, чем в первом. Вот еще пример: в OpenGL можно указывать каждую точку отдельно, а в OpenGL ES задается только серия точек одновременно. Однако зачастую можно имитировать элементы, отсутствующие в OpenGL ES, при помощи других элементов, более примитивных. Например, можно нарисовать прямоугольник, сложив два треугольника.

В OpenGL для рисования применяются два основных метода:

- `glVertexPointer`;
- `glDrawElements`.

ПРИМЕЧАНИЕ

В контексте OpenGL ES термины «API» и «метод» являются синонимами.

Метод `glVertexPointer` используется для указания серии точек (или вершин), а `glDrawElements` применяется для рисования элементов при помощи примитивных контуров, описанных выше. Мы рассмотрим эти методы подробнее, но сначала изучим номенклатуру, применяемую в наименованиях API OpenGL.

Все названия API в OpenGL начинаются с `gl`. За `gl` следует название метода. После названия метода может идти число, например 3, которое указывает либо на количество измерений (*x*, *y*, *z*), либо на количество аргументов. За названием метода указывается тип данных, например `f` для числа с плавающей точкой. (Обратитесь к онлайн-ресурсам по OpenGL и изучите различные типы данных и соответствующие им буквы.)

Следует упомянуть еще об одном условии. Если в качестве аргумента метод может принимать или байт (`b`) или число с плавающей точкой (`f`), то у метода будет два имени: одно будет начинаться с `b`, а другое — с `f`.

Теперь изучим оба метода рисования подробнее. Начнем с `glVertexPointer`.

glVertexPointer и указание вершин при рисовании

Метод `glVertexPointer` предназначен для определения массива точек, которые следует нарисовать. Каждая точка описывается в трех измерениях, то есть имеет три значения координат: *x*, *y* и *z*. В листинге 10.1 показано, как обозначить три точки в массиве.

Листинг 10.1. Пример определения координат вершин треугольника, нарисованного в OpenGL

```
float[] coords = {
   -0.5f, -0.5f, 0,       //p1: (x1,y1,z1)
    0.5f, -0.5f, 0,       //p2: (x1,y1,z1)
    0.0f,  0.5f, 0    //p3: (x1,y1,z1)
};
```

Структура, представленная в листинге 10.1, — это соприкасающиеся множества чисел с плавающей точкой, которые содержатся в соответствующем массиве, основанном на Java. Пока не волнуйтесь о том, как написать или скомпилировать этот код, — на данном этапе нам достаточно просто составить представление о том, как работают эти методы OpenGL ES. Ниже мы напишем специальное тестовое приложение, предназначенное для рисования простых фигур, и при работе над ним покажем рабочие примеры и код.

При изучении листинга 10.1 может возникнуть вопрос — какие единицы используются с координатами точек p1, p2 и p3. Если коротко, то при моделировании в трехмерном пространстве координатные единицы могут быть любыми, на ваш выбор. Но в дальнейшем может понадобиться задать *ограничивающий объем* (bounding volume), который еще называется *ограничивающим блоком* (bounding box), — в нем координаты переводятся в количественное выражение.

Например, ограничивающий блок может иметь форму куба с 5- или 2-дюймовыми ребрами. Такие координаты также называются *внешними*, так как представляют ваше виртуальное пространство независимо от ограничений, действующих в устройстве. Эти координаты будут более подробно объяснены в подразделе «Камера и координаты в OpenGL». Пока просто предположим, что используем куб, имеющий размер 2 единицы по всем измерениям с центром в точке ($x = 0$, $y = 0$, $z = 0$).

ПРИМЕЧАНИЕ

Термины «ограничивающий объем», «ограничивающий блок», «зона видимости», «поле видимости» и «видимое пространство» означают один и тот же феномен: трехмерный объем, имеющий форму пирамиды и определяющий, что именно видно на экране. Подробнее об этом будет рассказано в пункте «glFrustum и зона видимости» в подразделе «Камера и координаты в OpenGL».

Представьте, что начало координат находится в центре дисплея. Ось *Z* будет иметь отрицательные значения, уходя вглубь дисплея (в противоположную сторону от вас), и положительные, выходя из дисплея (по направлению к вам). Ось *X* будет иметь положительные значения по направлению вправо и отрицательные по направлению влево. Однако координаты также будут зависеть от направления, с которого вы просматриваете сцену.

Чтобы нарисовать эти точки в листинге 10.1, их нужно передать OpenGL ES при помощи метода `glVertexPointer`. Но ради повышения эффективности `glVertexPointer` принимает языково-независимый нативный буфер, а не массив чисел с плавающей точкой. В данном случае необходимо преобразовать массив чисел с плавающей точкой, основанный на Java, в подходящий C-подобный нативный буфер. Для такого преобразования потребуется использовать классы `java.nio`. В листинге 10.2 показан пример использования `nio`-буферов.

Листинг 10.2. Создание nio-буферов для чисел с плавающей точкой

```
jva.nio.ByteBuffer vbb = java.nio.ByteBuffer.allocateDirect(3 * 3 * 4);
vbb.order(ByteOrder.nativeOrder());
java.nio.FloatBuffer mFVertexBuffer = vbb.asFloatBuffer();
```

В листинге 10.2 байтовый буфер — это буфер памяти, разбитый на байты. Каждая точка имеет три координаты с плавающей точкой, так как в пространстве три оси. Каждое число, обозначающее координату, равно 4 байт. Следовательно, имеем 3 · 4 байт на каждую точку. Кроме того, треугольник имеет три точки. Соответственно, чтобы содержать информацию обо всех трех плавающих точках треугольника, нужно 3 · 3 · 4 байт.

Когда все точки будут собраны в нативный буфер, можно вызвать glVertexPointer, как это показано в листинге 10.3.

Листинг 10.3. Определение API glVertexPointer

```
glVertexPointer(
    // Если мы используем (x,y) или (x,y,z) с каждой из точек
    3,
    // каждое значение в буфере — это число с плавающей точкой.
    GL10.GL_FLOAT,
    // никакого расстояния между двумя точками нет
    0,
    // указание на начало буфера
    mFVertexBuffer);
```

Давайте рассмотрим на примере этого листинга аргументы метода glVertexPointer. Первый аргумент сообщает OpenGL ES, сколько параметров имеет точка или вершина. В данном случае мы указали три — x, y и z. Вы также можете задать два значения — только для x и y. В таком случае z будет равно нулю. Обратите внимание и на то, что первый аргумент — это не количество точек в буфере, а количество используемых измерений. То есть если передать 20 точек, чтобы нарисовать несколько треугольников, то в качестве первого аргумента будет передаваться не 20, а 2 или 3 — в зависимости от того, сколько измерений используется.

Второй аргумент указывает координаты, которые должны быть интерпретированы как числа с плавающей точкой. Третий аргумент, называемый stride, указывает число байт, которые находятся между двумя отдельными точками. В данном случае третий аргумент равен нулю, так как точки следуют непосредственно одна за другой. Иногда в буфер после каждой точки можно добавить цветовые атрибуты. Если вы собираетесь так сделать, то stride будет использоваться для пропуска этих атрибутов при спецификации вершин. Последний аргумент — это указатель на буфер, содержащий точки.

Теперь мы знаем, как задавать массив точек, которые должны быть нарисованы. Научимся рисовать такой массив точек, используя метод glDrawElements.

glDrawElements

После указания серии точек при помощи glVertexPointer используется метод glDrawElements для объединения этих точек в один из примитивных контуров, применяемых в OpenGL ES. Обратите внимание, что OpenGL — это машина состояний

(state machine). Она запоминает значения, заданные одним методом, когда вызывает другой метод, и т. д. с накоплением. То есть вам не требуется специально передавать точки, заданные методом glVertexPointer, методу glDrawElements. Он и так будет использовать эти точки. В листинге 10.4 показан пример этого метода с возможными аргументами.

Листинг 10.4. Пример glDrawElements

```
glDrawElements(
        // тип контура
        GL10.GL_TRIANGLE_STRIP,
        // количество индексов
        3,
        // размер каждого индекса
        GL10.GL_UNSIGNED_SHORT,
        // буфер, содержащий три индекса
        mIndexBuffer);
```

Первый аргумент указывает тип рисуемого геометрического контура: GL_TRIANGLE_STRIP представляющего собой полосу треугольников (множество связанных треугольников). Кроме того, этот аргумент может иметь следующие параметры: только точки (GL_POINTS), ломаные линии (GL_LINE_STRIP), только линии (GL_LINES), замкнутые ломаные линии (GL_LINE_LOOP), только треугольники (GL_TRIANGLES) и веера треугольников (GL_TRIANGLE_FAN).

Концепция STRIP в GL_LINE_STRIP и GL_TRIANGLE_STRIP предназначена для добавления новых точек с одновременным использованием имеющихся. Таким образом, мы избегаем необходимости заново указывать все точки в каждом новом объекте. Например, если указать в массиве четыре точки, то можно использовать полосы, чтобы нарисовать первый треугольник из (1,2,3), а второй — из (2,3,4). С каждой новой точкой добавляется новый треугольник (в «Красной книге» OpenGL этот момент описан более подробно). Данные параметры также можно варьировать, чтобы посмотреть, как именно рисуются треугольники при добавлении новых точек.

Концепция FAN в GL_TRIANGLE_FAN применяется к треугольникам, в которых начальная точка первого треугольника является начальной точкой и для всех остальных. Таким образом, у вас получается веерообразный или круглый объект, в центре которого находится первая вершина. Предположим, у нас в массиве шесть точек: (1,2,3,4,5,6). При применении FAN рисуются треугольники с вершинами (1,2,3), (1,3,4), (1,4,5) и (1,5,6). Каждая новая точка добавляет дополнительный треугольник, этот процесс можно сравнить с раскрыванием веера или развертыванием карточной колоды.

Остальные аргументы glDrawElements указывают, можно ли в данном случае многократно использовать спецификации точек. Например, в квадрате четыре точки. Любой квадрат может быть нарисован как комбинация двух треугольников. Если мы собираемся нарисовать два треугольника и составить из них квадрат — надо ли указывать шесть точек? Можно задать только четыре точки, но обратиться к ним шесть раз, чтобы нарисовать два треугольника. Этот процесс называется *индексированием в точечный буфер*.

Вот пример:

```
Points: (p1, p2, p3, p4)
Draw indices (p1, p2, p3,   p2,p3,p4)
```

Обратите внимание: в состав первого треугольника входят p1,p2,p3, а в состав второго — p2,p3,p4. При наличии этой информации второй аргумент glDrawElements указывает, сколько индексов будет находиться в индексном буфере.

Третий аргумент glDrawElements (см. листинг 10.4) указывает тип значений, содержащихся в массиве индексов. Значения могут относиться к одному из двух типов: беззнаковое целое (GL_UNSIGNED_SHORT) или беззнаковый байт (GL_UNSIGNED_BYTE).

Последний аргумент glDrawElements указывает на буфер индексов. Заполнение буфера индексов похоже на заполнение буфера вершин. Начинаем с массива Java и применяем пакет java.nio для преобразования этого массива в нативный буфер.

В листинге 10.5 приведен пример кода, в котором массив целых чисел {0,1,2} преобразуется в нативный буфер, подходящий для передачи в glDrawElements.

Листинг 10.5. Преобразование массива Java в буферы NIO

```
// определяется, как будут расположены точки.
short[] myIndecesArray = {0,1,2};

// получение буфера целых чисел
java.nio.ShortBuffer mIndexBuffer;

// выделение по 2 байта для значения каждого индекса
ByteBuffer ibb = ByteBuffer.allocateDirect(3 * 2);
ibb.order(ByteOrder.nativeOrder());
mIndexBuffer = ibb.asShortBuffer();

// данные, заполняющие буфер
for (int i=0;i<3;i++)
{
   mIndexBuffer.put(myIndecesArray[i]);
}
```

В этом фрагменте кода мы видели mIndexBuffer в действии. Теперь можно вернуться к листингу 10.4, чтобы лучше понять, как создается буфер индексов и как им управлять.

ПРИМЕЧАНИЕ

Не создавая новых точек, индексный буфер просто индексирует массив точек, указанных методом glVertexPointer. Это возможно, так как OpenGL запоминает наборы, заданные при предыдущих вызовах, сохраняя эти данные.

Теперь рассмотрим еще два часто используемых метода OpenGL ES: glClear и glColor. Оба этих метода будут применяться в тестовом приложении, к написанию которого мы вскоре перейдем.

glClear

Метод `glClear` используется для очистки поверхности изображения (drawing surface). Он позволяет сбросить не только цвет, но и глубину и тип использованных трафаретов (stencils). Чтобы указать элемент, который необходимо стереть, передается соответствующая константа: `GL_COLOR_BUFFER_BIT`, `GL_DEPTH_BUFFER_BIT` или `GL_STENCIL_BUFFER_BIT`.

Цветовой буфер отвечает за то, какие пикселы будут видимы, поэтому очистка этого буфера приводит к удалению с поверхности изображения всех цветов. Буфер глубины относится ко всем пикселам, видимым в трехмерной перспективе, в зависимости от того, насколько близко или далеко находится объект от наблюдателя. Буфер трафаретов (stencil buffer) несколько сложен для этого раздела, поэтому опишем его кратко: он применяется для создания визуальных эффектов на базе определенных динамических критериев. Для его очистки применяется `glClear`.

ПРИМЕЧАНИЕ

Трафарет — это рисовальный шаблон, который можно использовать для многократного воспроизведения рисунка. Например, при работе с Microsoft Office Visio все шаблоны рисунков, сохраняемые в файлах с расширением VSS, являются трафаретами. При обычном рисовании, не связанном с компьютером, трафарет создается вырезанием узоров в листе бумаги или другого плоского материала. Можно наложить краску поверх этого листа, а после его удаления получится копия вырезанного контура.

В нашем примере можно использовать для очистки буфера цвета следующий код:

```
// убираем с поверхности все цвета
gl.glClear(gl.GL_COLOR_BUFFER_BIT);
```

Теперь рассмотрим прикрепление заданного стандартного цвета к рисуемому объекту.

glColor

`glColor` используется для указания стандартного цвета для рисунка, который предстоит создать. В следующем фрагменте кода метод `glColor4f` устанавливает в качестве стандартного красный цвет:

```
// установка цвета для конкретного примера
glColor4f(1.0f, 0, 0, 0.5f);
```

Вспомним наш разговор о номенклатуре методов: `4f` относится к четырем аргументам, принимаемым методом, причем все аргументы — это числа с плавающей точкой. Четыре аргумента — это компоненты красного, зеленого, синего и альфа (градиент цвета). Начальные значения для каждого компонента равны 1 (`1,1,1,1`). В данном случае мы задаем красный цвет с половинным градиентом (который указывается в последнем аргументе альфа).

Итак, мы изучили базовые API, применяемые при рисовании, но нам еще нужно обратиться к нескольким вопросам, касающимся координат точек, которые указываются в трехмерном пространстве. В следующем подразделе объясняется,

как в OpenGL моделируется реальная перспектива с точки зрения наблюдателя, смотрящего в камеру.

Камера и координаты в OpenGL

Рисуя в трехмерном пространстве, вы в конечном итоге должны спроецировать трехмерное изображение на двухмерную поверхность — это напоминает работу с фотоаппаратом в реальном мире, где вы фотографируете трехмерное пространство и получаете двухмерный снимок. Этот символизм формально признан в OpenGL, и многие ее концепции объясняются по аналогии с камерой или фотоаппаратом.

В данном разделе будет рассказано, что от расположения камеры, направления объектива, ориентации камеры (например, вверх ногами), степени масштабирования и размера съемочной «пленки» зависит, какая часть нашего рисунка будет видимой.

Эти аспекты проецирования трехмерного изображения на двухмерный экран контролируются в OpenGL тремя методами:

- `gluLookAt` — контролирует направление камеры;
- `glFrustum` — контролирует размер зоны видимости или степень увеличения/уменьшения;
- `glViewport` — контролирует размер экрана или размер «пленки» в камере.

Не усвоив свойств трех этих API, вы не сможете программировать в OpenGL. Нужно еще глубже изучить метафору камеры в данной среде, чтобы объяснить действие трех этих API на изображение, видимое на экране в OpenGL. Начнем с `gluLookAt`.

gluLookAt и метафора камеры

Предположим, вы отправились на прогулку и собираетесь фотографировать окружающий ландшафт — цветы, деревья, реки и горы. Вы пришли на луг: перспектива, которая разворачивается перед вами, аналогична той, которую вы будете рисовать в OpenGL. Рисунки могут быть большими, как горы, или маленькими, как цветы. Но они будут пропорциональны друг другу. Как мы указывали выше, координаты, которые будут использоваться в этих рисунках, называются ***внешними***. В этих координатах можно, например, нарисовать линию длиной 4 единицы по оси X, задав точки с координатами `(0,0,0)` и `(4,0,0)`.

Когда будете готовы фотографировать, найдите место, где следует установить треногу. Потом поставьте фотоаппарат на треногу. Местонахождение вашего фотоаппарата — не треноги, а именно фотоаппарата — становится для него точкой отсчета координат. Итак, нужно взять лист бумаги и нарисовать это место, которое будет называться «точка обзора». Если не указать точку обзора, камера будет расположена в точке с координатами `(0,0,0)`, то есть точно в центре экрана. Обычно бывает нужно выбрать другую точку, чтобы можно было видеть плоскость (x, y) с координатой $z = 0$. Предположим, что камера находится в точке `(0,0,5)`. В таком случае точка обзора отодвинется от плоскости экрана на 5 единиц вперед.

На рис. 10.1 показано расположение камеры.

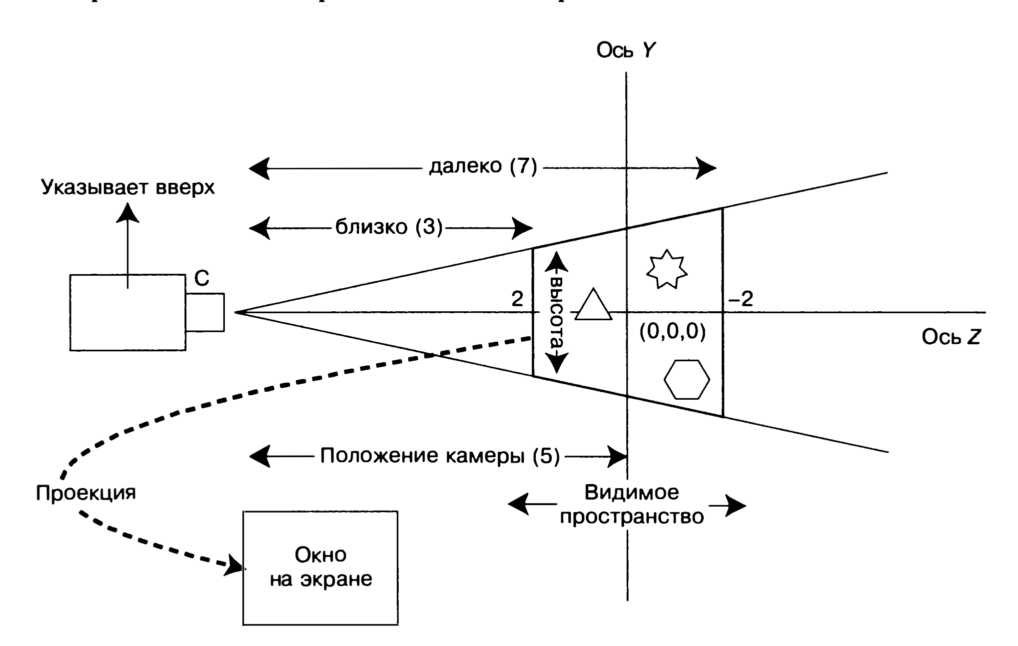


Рис. 10.1. Феномены OpenGL, связанные с просмотром; используется аналогия с камерой

Разместив камеру, смотрим вперед и определяем, какая часть сцены должна попасть в кадр. Камера размещается так, чтобы направление объектива совпадало с направлением вашего взгляда. Отдаленная точка, в которую вы смотрите, называется *точкой наблюдения* (view point) или *направлением взгляда* (look-at point). На самом деле вы указываете не точку, а направление. То есть, если указать точку наблюдения с координатами (0,0,0), объектив камеры будет направлен вдоль по оси *Z* на расстоянии 5 единиц. В результате сама камера будет находиться в точке (0,0,5). Эта ситуация показана на рис. 10.1, где объектив камеры направлен по оси *Z*.

Теперь предположим, что в точке начала координат находится прямоугольное здание. Вы хотите просмотреть его не в книжной (вертикальной), а в альбомной (горизонтальной) ориентации. Что делать? Как правило, можно оставить камеру в том же месте, чтобы она была направлена на точку начала координат, но повернуть аппарат на 90°. Речь в данном случае идет об *ориентации* камеры, так как камера зафиксирована в конкретной точке обзора и направлена к определенной точке наблюдения. Такая ориентация называется «вектор, направленный вверх».

Этот вектор просто определяет ориентацию камеры — вверх, вниз, вправо, влево или под углом. Ориентация камеры и в этот раз указывается при помощи точки. Представьте линию, идущую из начала координат, то есть из точки с координатами (0,0,0), а не из того места, где находится камера. Угол, которому противолежит эта линия в трехмерном пространстве в начале координат, является ориентацией камеры.

Например, вектор камеры, направленный вверх, может обозначаться как (0,1,0) или даже (0,15,0). В обоих случаях эффект будет одинаковый. Точка (0,1,0) удалена от начала координат по оси *Y*, направление вверх. Это означает, что вы располагаете камеру вертикально. Если использовать точку (0,-1,0), камера будет расположена вверх дном. Но в обоих случаях камера так и останется в точке (0,0,5) и будет направлена в начало координат — в точку (0,0,0). Три эти координаты можно обобщить следующим образом:

- (0,0,5) — точка обзора (местонахождение камеры);
- (0,0,0) — точка наблюдения (направление, которое указывает камера);
- (0,1,0) — вектор, направленный вверх (независимо от того, куда направлена камера — вверх, вниз или наклонена).

Для указания этих точек (точки обзора, точки наблюдения, а также вектора, направленного вверх) используется метод gluLookAt:

```
gluLookAt(gl, 0,0,5,  0,0,0, 0,1,0);
```

Здесь применяются следующие аргументы: первый набор координат относится к точке обзора, второй — к точке наблюдения, третий — к вектору, направленному вверх относительно точки начала координат.

Теперь рассмотрим, что такое «зона видимости».

glFrustum и зона видимости

Вы, вероятно, заметили, что ни одна из точек, описывающих расположение камеры с применением метода gluLookAt, не связана с размером. Описываются только расположение, направление и ориентация. Но как сообщить камере, где находится точка фокусировки? Насколько отдален объект, который вы хотите снять? Какова ширина и длина области, которая должна попасть в кадр? Метод OpenGL glFrustum используется для очерчивания интересующей вас части панорамы.

Представьте себе, что часть панорамы заключена в рамку, которая называется *зоной видимости,* или *видимым пространством* (см. на рис. 10.1 область в центре, очерченную жирной линией). В кадре оказывается все, что находится внутри этой рамки, а остальная часть панорамы обрезается и игнорируется. Итак, как указывается зона видимости? Сначала определяется *ближайшая точка* (near point), то есть расстояние между объективом и началом рамки. Затем находим *дальнюю точку* (far point), то есть расстояние между объективом и концом рамки. Расстояние по оси *Z* между ближней и дальней точками — это глубина рамки. Если указать для ближней точки значение 50, а для дальней — 200, то все, что находится между ними, попадет в кадр и глубина рамки будет равна 150. Кроме того, будет нужно указать левую и правую стороны, а также нижнюю границу рамки вдоль воображаемого *луча*, соединяющего камеру с точкой наблюдения.

В OpenGL такую рамку можно представить одним из двух способов. Один из способов называется *перспективной проекцией*, при построении которой используется феномен видимого пространства (frustum), рассмотренный нами выше. Такая перспектива имитирует естественную функцию камеры, в том числе пирамидальную структуру, основанием которой служит дальний план, а камера при

этом находится в апексе. Ближний план «отсекает» вершину пирамиды, располагая видимую область между ближним и дальним планами.

Другой способ построения рамки связан с представлением ее в виде куба. Такой вариант называется *ортогональной проекцией*. Она предназначена для геометрических чертежей, на которых необходимо сохранять реальные расстояния, независимо от удаленности от камеры.

В листинге 10.6 показано, как указать зону видимости для нашего примера.

Листинг 10.6. Указание зоны видимости методом glFrustum

```
// сначала рассчитываем характеристическое отношение
float ratio = (float) w / h;

// указываем, что собираемся использовать перспективную проекцию
glMatrixMode(GL10.GL_PROJECTION);

// указываем зону видимости: видимое пространство
gl.glFrustumf(
   -ratio,  // левый край поля видимости
   ratio,      // правый край поля видимости
   1,       // верхний край поля видимости
   -1,         // нижний край поля видимости
   3,       // насколько удалена от камеры передняя плоскость рамки
   7);         // насколько удалена от камеры задняя плоскость рамки
```

Поскольку в предыдущем фрагменте кода (листинг 10.6) для верхнего края установлено значение 1, а для нижнего -1, фронтальная высота рамки составит 2 единицы. Для указания размеров левой и правой сторон зоны видимости используются пропорциональные числа, получаемые с учетом характеристического отношения окна. Именно поэтому высота и ширина окна применяются в коде для составления пропорции. В коде также предполагается, что область действия будет иметь размер от 3 до 7 единиц по оси *Z*. Все, что будет начерчено за пределами этих координат (относительно камеры), будет невидимым.

Поскольку мы располагаем камеру в точке (0,0,5) и направляем к точке начала координат (0,0,0), 3 единицы вперед от камеры по направлению к началу координат соответствуют точке (0,0,2), а 7 единиц вперед от камеры по направлению к началу координат соответствуют точке (0,0,-2). Таким образом, плоскость, проходящая через начало координат, оказывается прямо в центре нашей трехмерной рамки.

Теперь мы определили точный размер нашей зоны видимости. Для перенесения (mapping) этих размеров на экран нужно разобрать еще один API — glViewport.

glViewport и размер экрана

При помощи glViewport указывается прямоугольная область на экране. На эту область будет проецироваться зона видимости. Данный метод принимает для описания прямоугольной рамки четыре аргумента: координаты *x* и *y* левого нижнего угла, за которыми следуют ширина и высота. В листинге 10.7 показано, как задать определенный экранный вид в качестве цели для этой проекции.

Листинг 10.7. Определение ViewPort через glViewPort

```
glViewport(0,      // координата "x" левого нижнего угла прямоугольника
                  // на экране
        0,        // координата "y" левого нижнего угла прямоугольника
                  // на экране
        width,    // ширина прямоугольника на экране
        height);// высота прямоугольника на экране
```

Если в нашем окне размер вида равен 100 пикселам в высоту, а высота зоны видимости составляет 10 единиц, то каждая логическая единица, равная 1 во внешних координатах, соответствует 10 пикселам в координатах экрана.

На данном этапе мы обсудили некоторые важные вводные концепции, связанные с OpenGL. В специальной книге по OpenGL этот материал может объясняться в нескольких отдельных главах. Эти фундаментальные феномены OpenDL нужно понимать и при написании кода OpenGL для Android. Имея такие предварительные знания, далее мы обсудим, как вызывать API OpenGL ES, изученные нами в этом разделе.

Взаимодействие OpenGL ES и Android

Как было сказано выше, OpenGL ES — это распространенный стандарт, поддерживаемый на многих платформах. Ядро стандарта составляет C-подобный API, обеспечивающий выполнение всей «рутинной» чертежной работы в OpenGL. Однако на каждой платформе и в каждой ОС по-своему внедрены дисплеи, буферы экрана и т. д. Специфичные аспекты обрисовываются и документируются в каждой отдельной операционной системе, и Android — не исключение.

В версии SDK 1.5 и выше в Android упрощены процессы взаимодействия и инициализации, связанные с рисованием в OpenGL. Поддержку этих механизмов обеспечивает пакет `android.opengl`. Класс, в котором содержится большая часть важных функций, называется `GLSurfaceView`. Он имеет внутренний интерфейс, называемый `GLSurfaceView.Renderer`. Очень важно знать два этих элемента, если вы хотите достичь значительных результатов при работе с OpenGL в Android.

Кроме того, в пакет включены следующие классы.

- **GLU.** В этот вспомогательный класс входят утилиты-обертки для основных API OpenGL ES, предназначенных для того, чтобы собрать вместе некоторые важные функции. Один из основных API, входящих в состав GLU, — это уже рассмотренный нами `gluLookAt`. Обратитесь к документации по API OpenGL SDK, где описаны другие подобные утилиты.
- **GLUtils.** В этом вспомогательном классе содержатся специфичные для Android утилиты, упрощающие взаимодействие с OpenGL. Основной метод из этого класса, которым мы будем пользоваться, называется `textImage2D`. Он берет точечный рисунок и открывает OpenGL доступ к нему для последующей текстуризации.
- **Matrix.** Это матрица преобразований, необходимая для таких операций, как масштабирование, перемещение и т. д.

- **Visibility.** Еще один вспомогательный класс; в этой главе мы не будем его использовать. Он работает с теми аспектами OpenGL, которые связаны с видимостью изображения, в частности определяет, какие именно участки треугольной сети (triangle mesh) будут видны на экране, а какие — нет.
- **GLDebugHelper.** Статический вспомогательный класс, позволяющий объединять в одной «обертке» интерфейсы GL и EGL, чтобы можно было контролировать журналирование, ошибки, дополнительные проверки и т. д.

Эти пакеты OpenGL поддерживают следующие интерфейсы.

- `GLSurfaceView.Renderer` — позволяет использовать при рисовании производные классы. Он дает возможность `GLSurfaceView` вызывать рисовальный элемент при изменении поверхности. Обычно программисты работают именно с этим интерфейсом. На его примере видно, как в Android делается попытка разделить рисование как таковое (оно осуществляется по общим правилам) и настройку OpenGL (она является специфичной для Android).
- `GLSurfaceView.EGLConfigChooser` — при помощи этого интерфейса `GLSurfaceView` может выбирать нужный объект `EGLConfig` для инициализации OpenGL. `EGLConfig` сообщает OpenGL тип характеристик отображения. В SDK версий ниже 1.5 приходилось самостоятельно организовывать взаимодействие этих классов. В SDK 1.5 и выше автоматически конфигурируются заданные по умолчанию параметры, если вы специально не укажете иные параметры.
- `GLSurfaceView.GLWrapper` — позволяет охватывать (wrap) интерфейс `gl` так, чтобы можно было перехватывать вызовы OpenGL, идущие по всей системе.

Использование GLSurfaceView и связанных классов

Начиная с версии SDK 1.5 и выше, общие принципы работы с OpenGL сильно упростились (в первом издании этой книги описано, как этот вопрос решается в Android 1.0). Ниже описаны типичные этапы использования этих классов при рисовании.

1. Внедряется интерфейс `Renderer`.
2. Проставляются настройки `Camera`, необходимые для рисования с применением данного варианта рендерера.
3. Задается код, описывающий рисунок, в методе `onDrawFrame` данного варианта.
4. Создается `GLSurfaceView`.
5. Настраивается рендерер, внедренный в `GLSurfaceView`.
6. Указывается, будет ли присутствовать в `GLSurfaceView` анимация.
7. `GLSurfaceView` задается в качестве вида-содержимого определенного явления. Этот вид можно использовать в любой части программы как самый обычный вид.

В листинге 10.8 показано типичное явление, в котором выполняются некоторые из описанных шагов.

Листинг 10.8. Простое тестовое явление с использованием OpenGL

```
public class OpenGLTestHarnessActivity extends Activity {
   private GLSurfaceView mTestHarness;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      mTestHarness = new GLSurfaceView(this);
      mTestHarness.setEGLConfigChooser(false);
      mTestHarness.setRenderer(new SimpleTriangleRenderer(this));
      mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
      //mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
      setContentView(mTestHarness);

   }
   @Override
   protected void onResume() {
      super.onResume();
      mTestHarness.onResume();
   }
   @Override
   protected void onPause() {
      super.onPause();
      mTestHarness.onPause();
   }
}
```

Теперь необходимо объяснить некоторые ключевые элементы, присутствующие в исходном коде. В следующей строке приведен фрагмент, инстанцирующий GLSurfaceView:

```
mTestHarness = new GLSurfaceView(this);
```

Затем вы сообщаете виду, что вам не требуется специальный элемент для выбора конфигурации EGL (config chooser). В таком случае по умолчанию будет выполнено следующее действие:

```
mTestHarness.setEGLConfigChooser(false);
```

Потом рендерер задается следующим образом:

```
mTestHarness.setRenderer(new SimpleTriangleRenderer(this));
```

Мы еще не показали SimpleTriangleRenderer, но это такой экземпляр интерфейса Renderer, в котором определен механизм рисования простого треугольника (скоро мы рассмотрим этот аспект).

Далее используется один из следующих двух методов, выбор метода влияет на то, будет или нет отображаться анимация.

```
mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
//mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
```

При выборе первой строки рисунок будет вызываться только однажды, точнее, в тот момент, когда его потребуется отобразить. Если выбрать второй вариант, рисовальный код будет вызываться многократно и рисунки можно будет анимировать.

Итак, мы изучили самые важные вопросы, связанные с взаимодействием OpenGL и Android. Далее мы расскажем о SimpleTriangleRender и покажем простую программу для его тестирования.

Простая тестовая программа, при помощи которой рисуется треугольник

Выше мы говорили, что при рисовании в OpenGL в систему внедряется интерфейс Renderer. Шаблон этого интерфейса показан в листинге 10.9.

Листинг 10.9. Интерфейс Renderer

```
public static interface GLSurfaceView.Renderer
{
   void onDrawFrame(GL10 gl);
   void onSurfaceChanged(GL10 gl, int width, int height);
   void onSurfaceCreated(GL10 gl, EGLConfig config);
}
```

Основная чертежная работа выполняется в методе onDrawFrame(). Всякий раз, когда для конкретного вида создается новая поверхность, вызывается метод onSurfaceCreated(), после чего можно вызвать ряд API OpenGL, предназначенных, в частности, для передачи полутонов (dithering), управления глубиной или других целей. Все эти API можно вызывать вне onDrawFrame().

Сходным образом при изменении поверхности из-за корректировки значений высоты и ширины окна вызывается метод onSurfaceChanged(). В нем можно настраивать камеру и зону видимости.

Даже в методе onDrawFrame() содержится немало вещей, которые вам регулярно придется применять при рисовании. Можно пользоваться этой унифицированностью и абстрагировать нужные методы на новом уровне, называемом AbstractRenderer, где будет использоваться всего один метод, который мы пока не внедрили, — draw().

В листинге 10.10 показан код AbstractRenderer:

Листинг 10.10. AbstractRenderer

```
// имя файла: AbstractRenderer.java
public abstract class AbstractRenderer
implements GLSurfaceView.Renderer
{
   public void onSurfaceCreated(GL10 gl, EGLConfig eglConfig) {
      gl.glDisable(GL10.GL_DITHER);
      gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT,
           GL10.GL_FASTEST);
      gl.glClearColor(.5f, .5f, .5f, 1);
      gl.glShadeModel(GL10.GL_SMOOTH);
```

```
        gl.glEnable(GL10.GL_DEPTH_TEST);
    }

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        gl.glViewport(0, 0, w, h);
        float ratio = (float) w / h;
        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);
    }

    public void onDrawFrame(GL10 gl)
    {
        gl.glDisable(GL10.GL_DITHER);
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);
        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        draw(gl);
    }
    protected abstract void draw(GL10 gl);
}
```

Иметь этот абстрактный класс очень полезно, так как он позволяет сосредоточиться на работе с методами, предназначенными только для рисования. Мы используем этот класс для создания собственного класса SimpleTriangleRenderer. В листинге 10.11 дан исходный код SimpleTriangleRenderer.

Листинг 10.11. SimpleTriangleRenderer

```
// имя файла: SimpleTriangleRenderer.java
public class SimpleTriangleRenderer extends AbstractRenderer
{
    // количество точек или вершин, которые мы собираемся использовать
    private final static int VERTS = 3;

    // необработанный нативный буфер, содержащий координаты точек
    private FloatBuffer mFVertexBuffer;

    // необработанный нативный буфер для содержания индексов,
    // обеспечивающих многократное использование точек
    private ShortBuffer mIndexBuffer;

    public SimpleTriangleRenderer(Context context)
    {
        ByteBuffer vbb = ByteBuffer.allocateDirect(VERTS * 3 * 4);
        vbb.order(ByteOrder.nativeOrder());
        mFVertexBuffer = vbb.asFloatBuffer();

        ByteBuffer ibb = ByteBuffer.allocateDirect(VERTS * 2);
```

```
        ibb.order(ByteOrder.nativeOrder());
        mIndexBuffer = ibb.asShortBuffer();

        float[] coords = {
                -0.5f, -0.5f, 0, // (x1,y1,z1)
                0.5f,  -0.5f, 0,
                0.0f,   0.5f, 0
        };
        for (int i = 0; i < VERTS; i++) {
            for(int j = 0; j < 3; j++) {
                mFVertexBuffer.put(coords[i*3+j]);
            }
        }
        short[] myIndecesArray = {0,1,2};
        for (int i=0;i<3;i++)
        {
            mIndexBuffer.put(myIndecesArray[i]);
        }
        mFVertexBuffer.position(0);
        mIndexBuffer.position(0);
    }

    // переопределенный метод
    protected void draw(GL10 gl)
    {
        gl.glColor4f(1.0f, 0, 0, 0.5f);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
        gl.glDrawElements(GL10.GL_TRIANGLES, VERTS,
                GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
    }
}
```

Хотя на первый взгляд и кажется, что в предыдущем примере очень много кода, большая его часть предназначена для определения вершин и последующей трансляции их из буферов Java в NIO-буферы. В самом методе draw всего три строки кода: для установления цвета, вершин и собственно для рисования.

ПРИМЕЧАНИЕ

Хотя мы и выделяем память для NIO-буферов, мы никогда не освобождаем их в коде. Так кто же их освобождает? Как эта память влияет на OpenGL?

Исследования, проведенные нами, показали, что пакет java.nio выделяет области памяти вне динамической памяти (хипа) Java, и эти области могут напрямую использоваться такими системами, как OpenGL, система файлового ввода/вывода и т. д. NIO-буферы, по существу, являются объектами Java, которые при определенных условиях указывают на нативный буфер. Такие объекты nio удаляются в процессе «сборки мусора». При «сборке мусора» они работают в опережающем режиме и удаляют нативную память. Сами программы Java не выполняют никаких специальных действий для очистки памяти.

Однако gc не запустится, если в динамической памяти Java будет недостаточно места. Это значит, что вы можете израсходовать всю нативную память, а gc этого не заметит. В Интернете есть примеры по этой теме, в которых исключение out of memory (память израсходована) запускает gc, а далее вы можете проверить, не освободилась ли память после активации gc.

При обычных обстоятельствах — в OpenGL это важно — можно распределять нативные буферы и не заниматься их освобождением специально — эту работу выполняет gc.

Теперь у нас готовы все три необходимых фрагмента и мы можем опробовать рисование. Наше явление показано в листинге 10.8. Абстрактный рендерер дан в листинге 10.10, а сам SimpleTriangleRenderer — в листинге 10.11. Теперь нам нужно просто активировать класс Activity из любого элемента меню при помощи следующего кода:

```
private void invokeSimpleTriangle()
{
   Intent intent = new Intent(this,OpenGLTestHarnessActivity.class);
   startActivity(intent);
}
```

Конечно, явление нужно будет зарегистрировать в файле описания Android:

```
<activity android:name=".OpenGLTestHarnessActivity"
   android:label="OpenGL 15 Test Harness"/>
```

После запуска этого кода у вас получится треугольник, как на рис. 10.2.

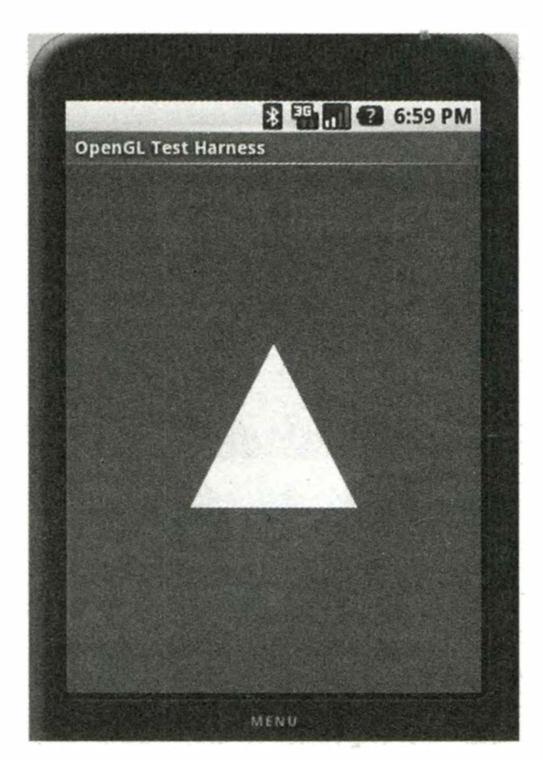


Рис. 10.2. Простой треугольник в OpenGL

Изменение настроек камеры

Чтобы лучше понимать работу с координатами в OpenGL, поэкспериментируем с методами, касающимися камеры, и посмотрим, как они воздействуют на треугольник, который изображен на рис. 10.2. Вспомним, где расположены вершины нашего треугольника: (-0.5,-0.5,0 0.5,-0.5,0 0,0.5,0). При применении к трем

этим точкам следующих трех методов AbstractRenderer (листинг 10.10) получается треугольник, показанный на рис. 10.2:

```
// Смотрим на экран (начало координат) с расстояния 5 единиц
// от поверхности экрана.
GLU.gluLookAt(gl, 0,0,5, 0,0,0, 0,1,0);

// Устанавливаем для высоты значение 2 единицы и для глубины 4 единицы.
gl.glFrustumf(-ratio, ratio, -1, 1, 3, 7);

// обычное содержимое окна
gl.glViewport(0, 0, w, h);
```

Теперь предположим, что нужно изменить вектор камеры, указывающий вверх, и направить его в отрицательном направлении по оси Y, вот так:

```
GLU.gluLookAt(gl, 0,0,5, 0,0,0, 0,-1,0);
```

Если это сделать, то получится треугольник, обращенный вершиной вниз (рис. 10.3). Если вы хотите внести такие изменения, можно найти в файле AbstractRenderer.java (см. листинг 10.10) метод, который необходимо изменить.

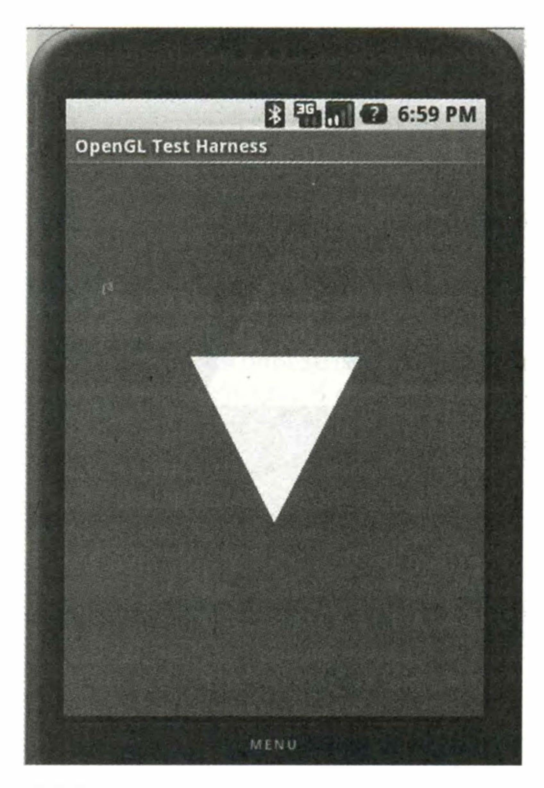


Рис. 10.3. Треугольник, видимый из перевернутой камеры

Сейчас посмотрим, что произойдет при изменении зоны видимости (также называемой областью видимости или рамкой). Следующий код позволяет увеличить

высоту и ширину поля видимости в 4 раза (чтобы лучше представить себе эти параметры, см. рис. 10.1). Как вы помните, четыре первых аргумента glFrustum указывают на передний прямоугольник поля видимости. Умножая каждое значение на 4, мы увеличиваем поле видимости в 4 раза:

```
gl.glFrustumf(-ratio * 4, ratio * 4, -1 * 4, 1 *4, 3, 7);
```

Этот код «сжимает» видимый треугольник, так как размеры треугольника остаются прежними, а поле видимости увеличивается. Вызов этого метода происходит в классе AbstractRenderer.java (см. листинг 10.10). На рис. 10.4 показано, что мы увидим после всех сделанных изменений.

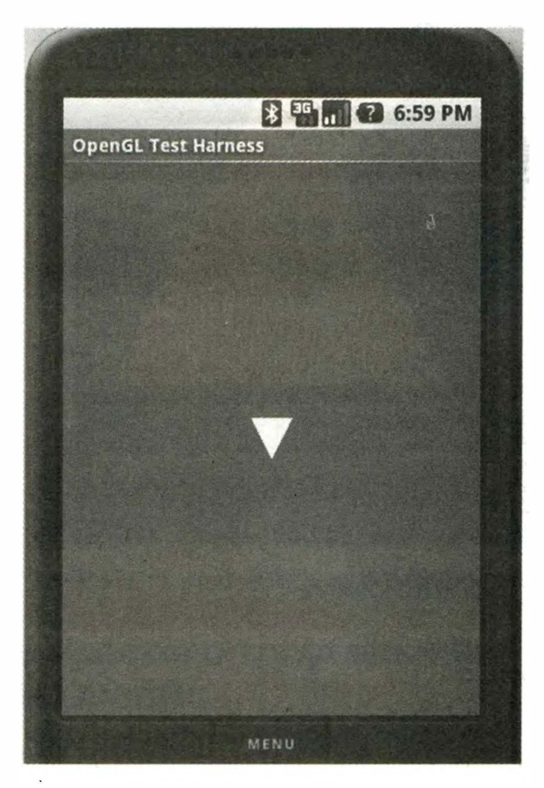


Рис. 10.4. Треугольник в четырехкратно увеличенном поле видимости

Использование индексов для добавления еще одного треугольника

Завершая работу с этими простыми треугольниками, мы покажем механизм наследования параметров класса AbstractRenderer и создадим еще один треугольник, просто добавив еще одну точку и воспользовавшись индексами. Необходимо задать четыре точки: (-1,-1, 1,-1, 0,1, 1,1). И мы скажем OpenGL начертить их как (0,1,2 0,2,3). В листинге 10.12 показан соответствующий код (обратите внимание: параметры треугольника изменены).

Листинг 10.12. Класс SimpleTriangleRenderer2

```
// имя файла: SimpleTriangleRenderer2.java
public class SimpleTriangleRenderer2 extends AbstractRenderer
{
   private final static int VERTS = 4;
   private FloatBuffer mFVertexBuffer;
   private ShortBuffer mIndexBuffer;

   public SimpleTriangleRenderer2(Context context)
   {
      ByteBuffer vbb = ByteBuffer.allocateDirect(VERTS * 3 * 4);
      vbb.order(ByteOrder.nativeOrder());
      mFVertexBuffer = vbb.asFloatBuffer();

      ByteBuffer ibb = ByteBuffer.allocateDirect(6 * 2);
      ibb.order(ByteOrder.nativeOrder());
      mIndexBuffer = ibb.asShortBuffer();

      float[] coords = {
            -1.0f, -1.0f, 0, // (x1,y1,z1)
             1.0f, -1.0f, 0,
             0.0f,  1.0f, 0,
             1.0f,  1.0f, 0
      };
      for (int i = 0; i < VERTS; i++) {
         for(int j = 0; j < 3; j++) {
            mFVertexBuffer.put(coords[i*3+j]);
         }
      }
      short[] myIndecesArray = {0,1,2, 0,2,3};
      for (int i=0;i<6;i++)
      {
         mIndexBuffer.put(myIndecesArray[i]);
      }
      mFVertexBuffer.position(0);
      mIndexBuffer.position(0);
   }

   protected void draw(GL10 gl)
   {
      gl.glColor4f(1.0f, 0, 0, 0.5f);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
      gl.glDrawElements(GL10.GL_TRIANGLES, 6, GL10.GL_UNSIGNED_SHORT,
            mIndexBuffer);
   }
}
```

Когда класс SimpleTriangleRenderer2 будет готов, можно изменить код OpenGLTestHarnessActivity (см. листинг 10.8),для активации этого рендерера, а не SimpleTriangleRenderer:

```
mTestHarness = new OpenGLTestHarness(this);
mTestHarness.setRenderer(new SimpleTriangleRenderer2(this));
```

Измененный фрагмент выделен полужирным. После внесения изменений в код можно снова запустить OpenGLTestHarnessActivity, чтобы увидеть два треугольника (рис. 10.5).

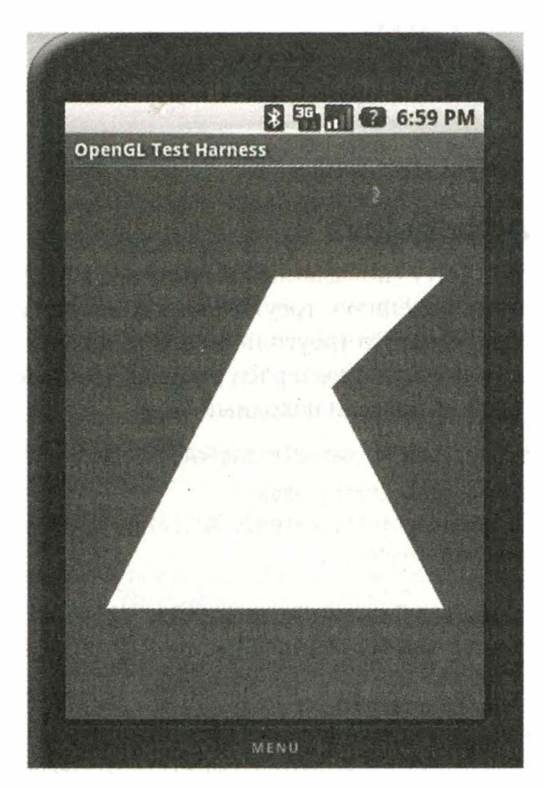


Рис. 10.5. Два треугольника, построенные при помощи четырех точек

Анимирование простого треугольника с применением OpenGL

Чтобы анимировать рисунок OpenGL, нужно просто изменить режим рендеринга объекта GLSurfaceView. В листинге 10.13 показан образец кода.

Листинг 10.13. Задание режима непрерывного рендеринга

```
// получение GLSurfaceView
GLSurfaceView openGLView;

// задание режима непрерывного рендеринга
openGLView.setRenderingMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
```

Обратите внимание: мы показываем здесь, как изменять режим рендеринга, поскольку в предыдущем разделе мы указали RENDERMODE_WHEN_DIRTY. Как мы уже говорили, RENDERMODE_CONTINUOUSLY является стандартной настройкой, поэтому анимация активирована по умолчанию. Когда задан режим непрерывного рендеринга, конкретный анимационный эффект обеспечивается методом рендерера onDraw.

Чтобы продемонстрировать этот феномен, мы покажем вам пример, в котором треугольник, нарисованный в предыдущем примере (см. листинг 10.11 и рис. 10.2) вращается. В этом примере используются два следующих файла:

- AnimatedTriangleActivity.java — обычное явление, в котором находится GLSurfaceView;
- AnimatedSimpleTriangleRenderer.java — файл, содержащий анимированный рисунок.

Давайте рассмотрим оба этих файла.

AnimatedTriangleActivity.java

Файл AnimatedTriangleActivity (показанный в листинге 10.14) соответствует простому неанимированному явлению с треугольником из листинга 10.8, в котором мы тестируем отрисовку простого треугольника. Это явление нужно нам в качестве поверхности, на которой будет вычерчен рисунок, а потом отображен на экране Android. В листинге 10.14 показан исходный код.

Листинг 10.14. Исходный код для AnimatedTriangleActivity

```
// имя файла: AnimatedTriangleActivity.java
public class AnimatedTriangleActivity extends Activity {
   private GLSurfaceView mTestHarness;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      mTestHarness = new GLSurfaceView(this);
      mTestHarness.setEGLConfigChooser(false);
      mTestHarness.setRenderer(new AnimatedSimpleTriangleRenderer(this));
      //mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
      setContentView(mTestHarness);
   }
   @Override
   protected void onResume() {
      super.onResume();
      mTestHarness.onResume();
   }
   @Override
   protected void onPause() {
      super.onPause();
      mTestHarness.onPause();
   }
}
```

Самая важная часть кода в этом листинге выделена полужирным шрифтом. Мы взяли предыдущее явление, которое использовали для создания простого рисунка (см. листинг 10.8) и исключили указание о режиме рендеринга, поставив перед ним знак комментария. Таким образом, GLSurfaceView переводится в стандартный режим

непрерывного рендеринга, в котором применяются многократные вызовы метода onDraw, относящегося к рендереру, в данном случае — к AnimatedSimpleTriangleRenderer.

Теперь рассмотрим строение класса AnimatedSimpleTriangleRenderer, который появляется в листинге 10.15. Он отвечает за рисование прямоугольников с короткими интервалами между появлениями рисунков; таким образом, имитируется анимация.

AnimatedSimpleTriangleRenderer

Класс AnimatedSimpleTriangleRenderer очень похож на SimpleTriangleRenderer (см. листинг 10.11), кроме того фрагмента, в котором описывается работа метода onDraw. Как видите, теперь в этом методе установлен поворот треугольника на заданный угол каждые четыре секунды. Поскольку изображение отрисовывается многократно, кажется, что треугольник медленно вращается. В листинге 10.15 дан полный код экземпляра класса AnimatedSimpleTriangleRenderer.

Листинг 10.15. Исходный код класса AnimatedSimpleTriangleRenderer

```
// имя файла AnimatedSimpleTriangleRenderer.java
public class AnimatedSimpleTriangleRenderer extends AbstractRenderer
{
   private int scale = 1;
   // количество точек или вершин, которые мы собираемся использовать
   private final static int VERTS = 3;

   // необработанный нативный буфер, в котором содержатся координаты точек
   private FloatBuffer mFVertexBuffer;

   // необработанный нативный буфер, в котором содержатся индексы
   //обеспечивающие многократное использование точек
   private ShortBuffer mIndexBuffer;

   public AnimatedSimpleTriangleRenderer(Context context)
   {
      ByteBuffer vbb = ByteBuffer.allocateDirect(VERTS * 3 * 4);
      vbb.order(ByteOrder.nativeOrder());
      mFVertexBuffer = vbb.asFloatBuffer();

      ByteBuffer ibb = ByteBuffer.allocateDirect(VERTS * 2);
      ibb.order(ByteOrder.nativeOrder());
      mIndexBuffer = ibb.asShortBuffer();

      float[] coords = {
            -0.5f, -0.5f, 0, // (x1,y1,z1)
             0.5f, -0.5f, 0,
             0.0f,  0.5f, 0
      };
      for (int i = 0; i < VERTS; i++) {
         for(int j = 0; j < 3; j++) {
```

```
            mFVertexBuffer.put(coords[i*3+j]);
         }
      }
      short[] myIndecesArray = {0,1,2};
      for (int i=0;i<3;i++)
      {
         mIndexBuffer.put(myIndecesArray[i]);
      }
      mFVertexBuffer.position(0);
      mIndexBuffer.position(0);
   }
   // переопределенный метод
   protected void draw(GL10 gl)
   {
      long time = SystemClock.uptimeMillis() % 4000L;
      float angle = 0.090f * ((int) time);

      gl.glRotatef(angle, 0, 0, 1.0f);

      gl.glColor4f(1.0f, 0, 0, 0.5f);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
      gl.glDrawElements(GL10.GL_TRIANGLES, VERTS,
         GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
   }
}
```

Теперь, когда у нас есть оба файла — AnimatedTriangleActivity.java и AnimatedSimpleTriangleRenderer.java, мы можем активировать анимированное явление из любого элемента меню, вызвав метод, показанный в листинге 10.16.

Листинг 10.16. Вызов анимированного явления

```
private void invoke15SimpleTriangle()
{
   Intent intent = new Intent(this,AnimatedTriangleActivity.class);
   startActivity(intent);

}
```

Не забудьте зарегистрировать явление в файле описания AndroidManifest.xml (листинг 10.17).

Листинг 10.17. Регистрация нового явления в файле AndroidManifest.xml

```
<activity android:name=".AnimatedTriangleActivity"
   android:label="OpenGL Animated Test Harness"/>
```

Бросаем вызов OpenGL: контуры и текстуры

Вы уже в основном изучили основы работы с OpenGL. Мы рассмотрели, как начертить простой треугольник, и в процессе решения этой задачи познакомились

с основами рисования. При помощи метафоры камеры мы описали систему координат и пояснили важность трех основных API: gluLookAt (настройка камеры), gluFrustum (настройка зоны видимости) и glViewPort (импорт зоны видимости на экран).

Опираясь на эти основные знания, мы сделали введение в стартовый фреймворк OpenGL в Android. Мы показали, как определяются базовые абстрактные классы, в которых можно инкапсулировать часто используемые, повторяющиеся настройки. При работе с этими абстрактными классами мы научились рисовать простой треугольник и анимировать его при помощи матриц преобразований.

Оставшаяся часть главы будет посвящена работе с более сложным уровнем OpenGL. В примерах, показанных выше, мы специально указывали вершины треугольника. Этот метод работы оказывается неудобным при рисовании квадратов, пятиугольников, шестиугольников и т. д. Далее мы покажем, что для рисования таких объектов нужны абстракции объектов более высокого уровня — в частности, контуры (shapes) и даже графы сцен (scene graph), причем для задания координат предназначаются именно контуры. Пользуясь такими методами, мы научимся рисовать многоугольники с любым количеством сторон и с любыми геометрическими характеристиками.

Затем мы займемся изучением текстур, применяемых в OpenGL. Текстуры позволяют прикреплять к поверхностям вычерчиваемых объектов точечные рисунки (bitmap) и любые другие картинки. Мы будем брать такие многоугольники, которые умеем рисовать, и прикреплять к ним рисунки. При этом мы освоим еще один важный аспект OpenGL — рисование нескольких фигур или контуров при помощи графического конвейера (pipeline).

После изучения этих фундаментальных моментов вам будет проще начать создавать рабочие трехмерные рисунки и сцены.

Простой прием работы с меню для демопримеров

До сих пор мы создавали отдельное явление для каждого рассматриваемого примера. При этом подразумевалось, что мы создаем явление для каждого демонстрационного примера и что каждое новое явление требуется регистрировать в XML-файле описания. Далее мы покажем прием, при помощи которого создается всего одно явление. Оно может в зависимости от выбранного элемента меню либо изменить вид, с которым оно связано, либо (в случае с GLSurfaceView) применять иной рендерер для каждого элемента меню.

Чтобы пояснить этот прием, сначала рассмотрим набор элементов меню, описывающих несколько демонстрационных элементов, с которыми нам доведется работать. Этот набор приведен в листинге 10.18. Основные фрагменты листинга выделены полужирным шрифтом, чтобы показать, какой элемент из набора мы собираемся рисовать при выборе того или иного пункта меню из общего набора.

Листинг 10.18. Структура меню для демонстрационных примеров OpenGL

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <!-- В этой группе используется стандартная категория. -->
   <group android:id="@+id/menuGroup_Main">

      <item android:id="@+id/mid_OpenGL_SimpleTriangle"
         android:title="Simple Triangle" />

      <item android:id="@+id/mid_OpenGL_AnimatedTriangle15"
         android:title="Animated Triangle" />

      <item android:id="@+id/mid_rectangle"
         android:title="rectangle" />

      <item android:id="@+id/mid_square_polygon"
         android:title="square polygon" />

      <item android:id="@+id/mid_polygon"
         android:title="polygon" />

      <item android:id="@+id/mid_textured_square"
         android:title="textured square" />

      <item android:id="@+id/mid_textured_polygon"
         android:title="textured polygon" />

      <item android:id="@+id/mid_OpenGL_Current"
         android:title="Current" />

      <item android:id="@+id/menu_clear"
         android:title="clear" />
   </group>
</menu>
```

Никакого волшебства тут не происходит. Эти меню указывают, что мы собираемся рисовать фигуры нескольких типов. Префикс mid означает ID элемента меню. Это просто условное обозначение, при помощи которого можно быстро находить ID элементов меню в Eclipse ADT. Для каждого элемента меню рисуется специальная сцена OpenGL. Элемент меню Simple Triangle (Простой треугольник) рисует простой треугольник, основываясь на специально указанных вершинах. Animated Triangle (Анимированный треугольник) берет такой простой треугольник и вращает его по мере прохождения времени. Элемент меню Rectangle (Прямоугольник) рисует прямоугольник, используя два треугольника, вершины которых специально указаны. В примере с многоугольником (Polygon) показан способ абстрактного определения многоугольника с применением его радиуса и количества сторон, с последующим генерированием вершин. Текстурированный квадрат (Textured Square) берет многоугольник-квадрат и прикрепляет к нему точечный

рисунок. В ходе создания текстурированного многоугольника (Textured Polygon) при помощи графического конвейера на самом деле рисуются два текстурированных многоугольника, причем заданная фигура дважды преобразуется и переводится в различные положения, чтобы можно было видеть два экземпляра одной и той же фигуры.

Посмотрим, как собрать из этих элементов меню единое явление. Вспомните листинг 10.8, где мы встречались с таким специальным явлением. Теперь сравним такое явление со следующим (листинг 10.19). В этом листинге дан полный код MultiViewTestHarnessActivity.

Листинг 10.19. MultiViewTestHarnessActivity

```
public class MultiViewTestHarnessActivity extends Activity {
   private GLSurfaceView mTestHarness;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      mTestHarness = new GLSurfaceView(this);
      mTestHarness.setEGLConfigChooser(false);

      Intent intent = getIntent();
      int mid = intent.getIntExtra("com.ai.menuid",
            R.id.MenuId_OpenGL15_Current);
      if (mid == R.id.MenuId_OpenGL15_Current)
      {
         mTestHarness.setRenderer(new TexturedPolygonRenderer(this));
         mTestHarness.setRenderMode(
            GLSurfaceView.RENDERMODE_CONTINUOUSLY);
         setContentView(mTestHarness);
         return;
      }

      if (mid == R.id.mid_OpenGL15_SimpleTriangle)
      {
         mTestHarness.setRenderer(new SimpleTriangleRenderer(this));
         mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
         setContentView(mTestHarness);
         return;
      }
      if (mid == R.id.mid_OpenGL15_AnimatedTriangle15)
      {
         mTestHarness.setRenderer(new
               AnimatedSimpleTriangleRenderer(this));
         setContentView(mTestHarness);
         return;
      }
      if (mid == R.id.mid_rectangle)
      {
```

```
            mTestHarness.setRenderer(new SimpleRectRenderer(this));
            mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
            setContentView(mTestHarness);
            return;
        }
        if (mid == R.id.mid_square_polygon)
        {
            mTestHarness.setRenderer(new SquareRenderer(this));
            mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
            setContentView(mTestHarness);
            return;
        }
        if (mid == R.id.mid_polygon)
        {
            mTestHarness.setRenderer(new PolygonRenderer(this));
            setContentView(mTestHarness);
            return;
        }
        if (mid == R.id.mid_textured_square)
        {
            mTestHarness.setRenderer(new TexturedSquareRenderer(this));
            mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
            setContentView(mTestHarness);
            return;
        }
        // в противном случае, сделать так
        mTestHarness.setRenderer(new TexturedPolygonRenderer(this));
        mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_CONTINUOUSLY);
        setContentView(mTestHarness);
        return;
    }
    @Override
    protected void onResume() {
        super.onResume();
        mTestHarness.onResume();
    }
    @Override
    protected void onPause() {
        super.onPause();
        mTestHarness.onPause();
    }
}
```

В листинге 10.19 в первую очередь необходимо обратить внимание на имя явления. Мы назвали его `MultiViewTestHarnessActivity`, указывая, что при активации это явление может быть несущим для нескольких видов, в зависимости от того, из какого меню оно активируется. Каким образом явление узнает, из какого именно элемента его активировали? Этот механизм показан в следующем фрагменте кода (листинг 10.20).

Листинг 10.20. Считывание ID элемента меню из намерения

```
Intent intent = getIntent();
int mid = intent.getIntExtra("com.ai.menuid", R.id.mid_OpenGL_Current);
if (mid == R.id.MenuId_OpenGL15_Current)
{
....
}
```

Первый вызов сообщает явлению, как именно оно было вызвано. Активатор передает ID меню с данными намерения extra. Вторая строка находит эти дополнительные данные, и если она не передается, то считается, что применено меню mid_OpenGL_Current, это значит, что вы должны просто выдать вид, который задан по умолчанию и которому не присвоен особый элемент меню.

В листинге 10.21 показано, как MultiViewTestHarnessActivity активируется другим основным явлением, которое может быть в данном примере ведущим элементом. Это основное явление затем начинает управлять меню и передавать события, связанные с активацией элементов меню, к MultiViewTestHarnessActivity.

Листинг 10.21. Передача ID меню вместе с намерением

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
   if (item.getItemId() == R.id.mid_OpenGL10_SimpleTriangle)
   {
   //..Направить данный элемент меню к основному явлению,
   // расположенному на локальном устройстве.
   //..Причем это явление может использоваться и в других целях.
      return true;
   }

   // Эти элементы меню должны быть направлены к multiview.
   this.invokeMultiView(item.getItemId());
   return true;
}


// здесь множественный вид запускается при помощи загруженного намерения,
// имеющего id меню
// mid: menu id
private void invokeMultiView(int mid)
{
   Intent intent = new Intent(this,MultiViewTestHarnessActivity.class);
   intent.putExtra("com.ai.menuid", mid);
   startActivity(intent);
}
```

Ради экономии места мы не приводим здесь код основного явления, поскольку оно практически не поможет вам лучше понять Android или OpenGL. Вышеуказанный

код должен дать вам представление о том, как вставить в программу явление для проведения любых тестов, которые вам могут понадобиться.

Основное явление, используемое в этой главе, будет выглядеть так, как на рис. 10.6.

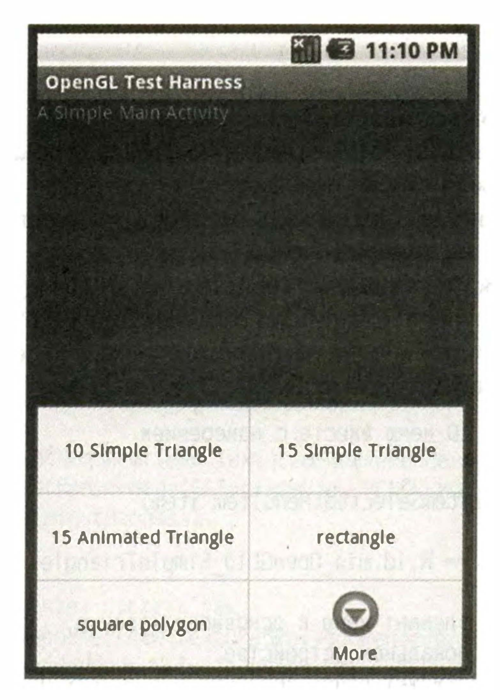


Рис. 10.6. Основное явление тестового приложения OpenGL

Как видите, это простое явление, в состав которого входит комплект меню. При активации любых меню каждое из них передается к MultiViewTestHarnessActivity (показанному в листинге 10.19).

При помощи такого метода мы можем заново внедрить в программу простой треугольник при помощи следующего многовидового явления:

```
if (mid == R.id.mid_OpenGL15_SimpleTriangle)
{
   mTestHarness.setRenderer(new SimpleTriangleRenderer(this));
   mTestHarness.setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
   setContentView(mTestHarness);
   return;
}
```

Обратите внимание: мы использовали тот же объект рендеринга и тем же способом, что при работе с SimpleTriangleActivity в листинге 10.8, но теперь этот объект реализован вместе с другими демонстрационными примерами (где мы рисовали более одной фигуры, сложнее, чем треугольник) в листинге 10.19. В этом образце также отражен принцип использования дополнительной информации, содержащейся в намерениях.

Рисование прямоугольника

Прежде чем перейти к изучению контуров, закрепим наше умение рисовать при помощи специально определяемых вершин и создадим прямоугольник, использовав два треугольника. Кроме того, мы выполним основную подготовительную работу для расширения прямоугольника до любого многоугольника.

У вас уже достаточно базовых знаний для того, чтобы рисовать простейшие треугольники, и теперь мы покажем код для рисования прямоугольника (листинг 10.22) и кратко его прокомментируем.

Листинг 10.22. Рендерер простого прямоугольника

```
public class SimpleRectangleRenderer extends AbstractRenderer
{
   // Количество точек или вершин, которые мы собираемся использовать.
   private final static int VERTS = 4;

   // Необработанный нативный буфер, в котором содержатся координаты точек
   private FloatBuffer mFVertexBuffer;

   // Необработанный нативный буфер, в котором содержатся индексы
   // обеспечивающие многократное использование точек.
   private ShortBuffer mIndexBuffer;

   public SimpleRectRenderer(Context context)
   {
      ByteBuffer vbb = ByteBuffer.allocateDirect(VERTS * 3 * 4);
      vbb.order(ByteOrder.nativeOrder());
      mFVertexBuffer = vbb.asFloatBuffer();

      ByteBuffer ibb = ByteBuffer.allocateDirect(6 * 2);
      ibb.order(ByteOrder.nativeOrder());
      mIndexBuffer = ibb.asShortBuffer();

      float[] coords = {
            -0.5f, -0.5f, 0, // (x1,y1,z1)
             0.5f, -0.5f, 0,
             0.5f,  0.5f, 0,
            -0.5f,  0.5f, 0,
      };

      for (int i = 0; i < VERTS; i++) {
         for(int j = 0; j < 3; j++) {
            mFVertexBuffer.put(coords[i*3+j]);
         }
      }
      short[] myIndecesArray = {0,1,2,0,2,3};
      for (int i=0;i<6;i++)
      {
         mIndexBuffer.put(myIndecesArray[i]);
      }
```

```
      mFVertexBuffer.position(0);
      mIndexBuffer.position(0);
   }

   // переопределенный метод
   protected void draw(GL10 gl)
   {
      RegularPolygon.test();
      gl.glColor4f(1.0f, 0, 0, 0.5f);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
      gl.glDrawElements(GL10.GL_TRIANGLES, 6,
         GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
   }
}
```

Обратите внимание: способ рисования прямоугольника очень напоминает рисование треугольника. Мы просто указываем не три вершины, а четыре. Затем мы применяем индексы следующим образом:

```
short[] myIndecesArray = {0,1,2,0,2,3};
```

Мы дважды использовали пронумерованные вершины (от 0 до 3) так, чтобы из каждых трех вершин получался треугольник. То есть вершины (0,1,2) относятся к первому треугольнику, а (0,2,3) — ко второму. Если нарисовать два этих треугольника при помощи базовых элементов GL_TRIANGLES, мы получим нужный нам прямоугольник.

Этот прямоугольник показан на рис. 10.7.

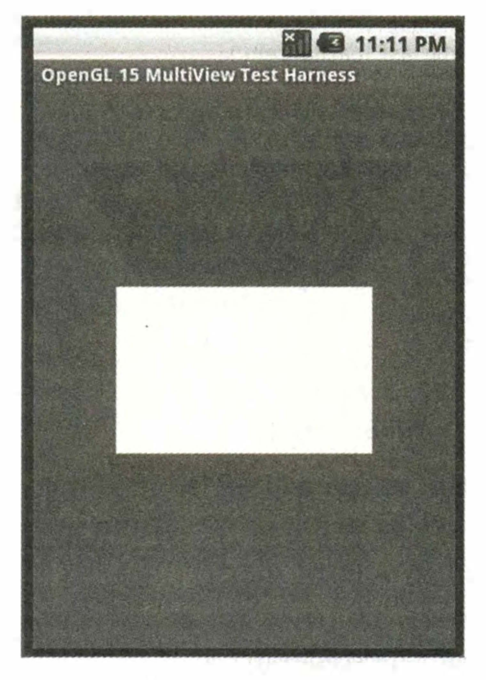


Рис. 10.7. Прямоугольник OpenGL, составленный из двух треугольников

Работа с контурами

Метод работы, требующий специально указывать все вершины, на практике достаточно утомителен. Например, чтобы нарисовать 20-сторонний многоугольник, понадобится указать 20 вершин, то есть 60 значений, так как каждая вершина описывается тремя значениями. Это совершенно непрактично.

Правильный многоугольник в качестве контура

При рисовании таких фигур, как треугольник или квадрат, целесообразнее определять абстрактный многоугольник путем указания нескольких его аспектов, в частности точки отсчета и радиуса, а затем программировать этот многоугольник так, чтобы он возвращал нам массив вершин и, возможно, массив индексов (чтобы мы могли рисовать отдельные треугольники). Таким образом, основная задача — создать абстрактный многоугольник, который будет выполнять эти задачи. Оформим его как класс RegularPolygon. Имея объект такого рода, его можно использовать так, как это показано в листинге 10.23, для отображения различных правильных многоугольников.

Листинг 10.23. Использование объекта RegularPolygon

```
// Многоугольник с 4 сторонами и радиусом 0.5,
// размещенный в точке (x,y,z) с координатами (0,0,0).
RegularPolygon square = new RegularPolygon(0,0,0,0.5f,4);

// многоугольник должен возвращать вершины
mFVertexBuffer = square.getVertexBuffer();

// многоугольник должен возвращать треугольники
mIndexBuffer = square.getIndexBuffer();

// вам это понадобится для glDrawElements
numOfIndices = square.getNumberOfIndices();

// устанавливаем буферы для начала работы
this.mFVertexBuffer.position(0);
this.mIndexBuffer.position(0);

// устанавливаем указатель вершин
gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);

// рисуем его с указанным количеством индексов
gl.glDrawElements(GL10.GL_TRIANGLES, numOfIndices,
   GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
```

Обратите внимание: мы получили нужные вершины и индексы из контура square. Хотя мы и не абстрагировали в виде базового контура идею получения вершин и индексов, возможна ситуация, в которой RegularPolygon можно будет выводить из базового контура, определяющего интерфейс для такого базового контракта. Ниже приведен соответствующий пример:

```
public interface Shape
{
```

```
    FloatBuffer        getVertexBuffer();
    ShortBuffer        getIndexBuffer();
    int            getNumberofIndices();
}
```

Идею определения базового интерфейса для контура мы оставляем вам как пищу для размышлений — возможно, она пригодится вам в ходе практической работы. Пока мы будем использовать эти методы непосредственно с RegularPolygon.

Реализация контура RegularPolygon

Как было указано выше, RegularPolygon отвечает за возвращение информации, необходимой для рисования правильного многоугольника в OpenGL, а именно данных о вершинах. Сначала нам потребуется механизм для определения того, что это за контур и какова его геометрия.

В случае с правильным многоугольником это можно сделать несколькими способами. Мы определяем правильный многоугольник, используя количество сторон и расстояние от центра многоугольника до одной из его вершин. Это расстояние мы назовем «радиус», так как вершины правильного многоугольника расположены по периметру круга, центр которого совпадает с центром правильного многоугольника. Итак, по радиусу такого круга и количеству сторон можно узнать, какой многоугольник мы собираемся рисовать. Зная координаты центра, мы также узнаем геометрию нашего многоугольника.

Класс RegularPolygon должен дать нам координаты всех вершин многоугольника, используя в качестве входных данных координаты его центра и радиус. Опять же это можно сделать несколькими способами. В зависимости от того, какой из математических методов вы изберете в итоге (вы должны знать их из курса средней или высшей школы), после получения вершин можно будет приступать к рисованию.

Ниже показан метод, использованный авторами книги. В основе его лежит допущение, что радиус равен единице. Далее были получены углы для линий, соединяющих каждую из вершин многоугольника с его центром. Эти углы были сохранены как массив. Для каждого угла была рассчитана проекция по оси X, эти проекции были собраны в массив множителей по оси x (массив множителей применялся потому, что исходной точкой послужила единица радиуса). Зная реальный радиус, мы умножаем эти величины на реальный радиус и получаем реальную координату по оси X. Затем полученные реальные координаты сохраняются в массиве, называемом массивом X. Аналогичная операция осуществляется и с проекциями по оси Y.

Теперь, когда мы разобрались, что должно происходить в экземпляре RegularPolygon, представляем вам исходный код, описывающий функции такого экземпляра. В листинге 10.24 обобщен весь код RegularPolygon (обратите внимание на то, что он занимает несколько страниц). Чтобы изучать этот код было не так трудно, мы выделили названия функций и вставили в код внутристрочные комментарии перед началом каждой функции.

Основные функции определяются в списке, следующем за листингом 10.24. Здесь важно обрисовать вершины и их возвращение. Если этот код окажется для

вас слишком непонятным, вам не составит труда написать собственный вариант кода для возвращения вершин.

Отметим также, что в приведенном коде есть функции, работающие с текстурированием. Эти функции будут подробно рассмотрены в подразделе «Работа с текстурами».

Листинг 10.24. Внедрение контура RegularPolygon

```
public class RegularPolygon
{
   // Пространство, в котором содержатся
   // координаты (x,y,z) центра: cx,cy,cz
   // и радиус "r".
   private float     cx, cy, cz, r;
   private int          sides;

   // массив координат: точки вершин (x,y)
   private float[] xarray = null;
   private float[] yarray = null;

   // массив текстур: точки (x,y) также называемые (s,t)
   // здесь идет фигура, соотносимая с точечным рисунком текстуры
   private float[] sarray = null;
   private float[] tarray = null;

   //**********************************************
   // Конструктор
   //**********************************************

   public RegularPolygon(float incx, float incy,
                    float incz, // (x,y,z) начало координат
                    float inr, // радиус
                    int insides) // количество сторон
   {
      cx = incx;
      cy = incy;
      cz = incz;
      r = inr;
      sides = insides;

      // выделение памяти для массивов
      xarray = new float[sides];
      yarray = new float[sides];

      // выделение памяти для массивов точек текстур
      sarray = new float[sides];
      tarray = new float[sides];

      // расчет точек вершин
      calcArrays();

      // расчет точек текстур
```

```
      calcTextureArrays();
   }

   //********************************************
   // Получение и конвертирование координат вершин
   // на базе данных о начале координат и радиусе.
   // Реальная логика углов, действующая в функциях getMultiplierArray().
   //********************************************
   private void calcArrays()
   {
      // Получение точек вершин с допущением,
      // что они расположены по кругу
      // с радиусом "1" и с центром в точке "ноль".
      float[] xmarray = this.getXMultiplierArray();
      float[] ymarray = this.getYMultiplierArray();

      // Расчет массива xarray: получение вершины
      // путем сложения с частью "x" начала координат.
      // Умножение координат на радиус (масштаб).
      for(int i=0;i<sides;i++)
      {
         float curm = xmarray[i];
         float xcoord = cx + r * curm;
         xarray[i] = xcoord;
      }
      this.printArray(xarray, "xarray");

      // расчет yarray: аналогичные операции с координатами по оси y
      for(int i=0;i<sides;i++)
      {
         float curm = ymarray[i];
         float ycoord = cy + r * curm;
         yarray[i] = ycoord;
      }
      this.printArray(yarray, "yarray");
   }

   //********************************************
   // Расчет массивов текстур.
   // См. раздел о текстурах, где эта проблема рассмотрена более подробно,
   // метод очень похож.
   // В данном случае многоугольник соотносится с пространством,
   // имеющим форму квадрата.
   //********************************************
   private void calcTextureArrays()
   {
      float[] xmarray = this.getXMultiplierArray();
      float[] ymarray = this.getYMultiplierArray();

      // расчет xarray
      for(int i=0;i<sides;i++)
```

```
    {
       float curm = xmarray[i];
       float xcoord = 0.5f + 0.5f * curm;
       sarray[i] = xcoord;
    }
    this.printArray(sarray, "sarray");

    // расчет yarray
    for(int i=0;i<sides;i++)
    {
       float curm = ymarray[i];
       float ycoord = 0.5f + 0.5f * curm;
       tarray[i] = ycoord;
    }
    this.printArray(tarray, "tarray");
}

//*********************************************
// Преобразование массива вершин java
// в буфер nio для чисел с плавающей точкой.
//*********************************************
public FloatBuffer getVertexBuffer()
{
    int vertices = sides + 1;
    int coordinates = 3;
    int floatsize = 4;
    int spacePerVertex = coordinates * floatsize;

    ByteBuffer vbb = ByteBuffer.allocateDirect(
                    spacePerVertex * vertices);
    vbb.order(ByteOrder.nativeOrder());
    FloatBuffer mFVertexBuffer = vbb.asFloatBuffer();

    // помещение первой координаты (x,y,z:0,0,0)
    mFVertexBuffer.put(cx); //x
    mFVertexBuffer.put(cy); //y
    mFVertexBuffer.put(0.0f); //z

    int totalPuts = 3;
    for (int i=0;i<sides;i++)
    {
       mFVertexBuffer.put(xarray[i]); //x
       mFVertexBuffer.put(yarray[i]); //y
       mFVertexBuffer.put(0.0f); //z
       totalPuts += 3;
    }
    Log.d("total puts:",Integer.toString(totalPuts));
    return mFVertexBuffer;
}

//*********************************************
// Преобразование буфера текстур в буфер nio.
```

```
//**********************************************
public FloatBuffer getTextureBuffer()
{
   int vertices = sides + 1;
   int coordinates = 2;
   int floatsize = 4;
   int spacePerVertex = coordinates * floatsize;

   ByteBuffer vbb = ByteBuffer.allocateDirect(
                  spacePerVertex * vertices);
   vbb.order(ByteOrder.nativeOrder());
   FloatBuffer mFTextureBuffer = vbb.asFloatBuffer();

   // помещение первой координаты (x,y (s,t):0,0)
   mFTextureBuffer.put(0.5f); // x или s
   mFTextureBuffer.put(0.5f); // y или t

   int totalPuts = 2;
   for (int i=0;i<sides;i++)
   {
      mFTextureBuffer.put(sarray[i]); // x
      mFTextureBuffer.put(tarray[i]); // y
         totalPuts += 2;
   }
   Log.d("total texture puts:",Integer.toString(totalPuts));
   return mFTextureBuffer;
}

//**********************************************
// Расчет индексов, формирующих несколько треугольников.
// Начало с центральной вершины, расположенной в точке 0,
// затем подсчет в направлении по часовой стрелке, то есть
// 0,1,2, 0,2,3, 0,3,4.и т. д.
//**********************************************
public ShortBuffer getIndexBuffer()
{
   short[] iarray = new short[sides * 3];
      ByteBuffer ibb = ByteBuffer.allocateDirect(sides * 3 * 2);
      ibb.order(ByteOrder.nativeOrder());
      ShortBuffer mIndexBuffer = ibb.asShortBuffer();
      for (int i=0;i<sides;i++)
      {
         short index1 = 0;
         short index2 = (short)(i+1);
         short index3 = (short)(i+2);
         if (index3 == sides+1)
         {
            index3 = 1;
         }
```

```
            mIndexBuffer.put(index1);
            mIndexBuffer.put(index2);
            mIndexBuffer.put(index3);

            iarray[i*3 + 0]=index1;
            iarray[i*3 + 1]=index2;
            iarray[i*3 + 2]=index3;
        }
        this.printShortArray(iarray, "index array");
        return mIndexBuffer;
    }

    //*********************************************
    // Здесь берется массив углов
    // для каждой вершины и рассчитывается их проекционный множитель
    // по оси X.
    //*********************************************
    private float[] getXMultiplierArray()
    {
        float[] angleArray = getAngleArrays();
        float[] xmultiplierArray = new float[sides];
        for(int i=0;i<angleArray.length;i++)
        {
            float curAngle = angleArray[i];
            float sinvalue = (float)Math.cos(Math.toRadians(curAngle));
            float absSinValue = Math.abs(sinvalue);
            if (isXPositiveQuadrant(curAngle))
            {
                sinvalue = absSinValue;
            }
            else
            {
                sinvalue = -absSinValue;
            }
            xmultiplierArray[i] = this.getApproxValue(sinvalue);
        }
        this.printArray(xmultiplierArray, "xmultiplierArray");
        return xmultiplierArray;
    }

    //*********************************************
    // Здесь берется массив углов
    // для каждой вершины и рассчитывается их проекционный множитель
    // по оси Y.
    //*********************************************
    private float[] getYMultiplierArray() {
        float[] angleArray = getAngleArrays();
        float[] ymultiplierArray = new float[sides];
        for(int i=0;i<angleArray.length;i++) {
```

```
        float curAngle = angleArray[i];
        float sinvalue = (float)Math.sin(Math.toRadians(curAngle));
        float absSinValue = Math.abs(sinvalue);
        if (isYPositiveQuadrant(curAngle)) {
            sinvalue = absSinValue;
        }
        else {
            sinvalue = -absSinValue;
        }
        ymultiplierArray[i] = this.getApproxValue(sinvalue);
    }
    this.printArray(ymultiplierArray, "ymultiplierArray");
    return ymultiplierArray;
}

//**************************************************
// Возможно, эта функция не понадобится.
// Протестируйте ее и сотрите, если окажется, что она вам не нужна.
//**************************************************
private boolean isXPositiveQuadrant(float angle) {
    if ((0 <= angle) && (angle <= 90)) { return true; }
    if ((angle < 0) && (angle >= -90)) { return true; }
    return false;
}

//**************************************************
// Возможно, эта функция не понадобится.
// Протестируйте ее и сотрите, если окажется, что она вам не нужна.
//**************************************************
private boolean isYPositiveQuadrant(float angle) {
    if ((0 <= angle) && (angle <= 90)) { return true; }
    if ((angle < 180) && (angle >= 90)) {return true;}
    return false;
}
//**************************************************
// Здесь рассчитываются углы для линий,
// идущих из центра к каждой из вершин.
//**************************************************
private float[] getAngleArrays() {
    float[] angleArray = new float[sides];
    float commonAngle = 360.0f/sides;
    float halfAngle = commonAngle/2.0f;
    float firstAngle = 360.0f - (90+halfAngle);
    angleArray[0] = firstAngle;

    float curAngle = firstAngle;
    for(int i=1;i<sides;i++)
    {
        float newAngle = curAngle - commonAngle;
```

```
            angleArray[i] = newAngle;
            curAngle = newAngle;
        }
        printArray(angleArray, "angleArray");
        return angleArray;
    }

    //**********************************************
    // здесь при необходимости производится округление
    //**********************************************
    private float getApproxValue(float f) {
        return (Math.abs(f) < 0.001) ? 0 : f;
    }
    //**********************************************
    // Возвращает столько индексов, сколько вам будет нужно
    // с учетом количества сторон.
    // Здесь рассчитывается необходимое количество треугольников
    // для создания многоугольника с умножением на 3.
    // Просто получается, что количество треугольников
    // равняется количеству сторон.
    //**********************************************
    public int getNumberOfIndices() {
        return sides * 3;
    }
    public static void test() {
        RegularPolygon triangle = new RegularPolygon(0,0,0,1,3);
    }
    private void printArray(float array[], String tag) {
        StringBuilder sb = new StringBuilder(tag);
        for(int i=0;i<array.length;i++) {
            sb.append(";").append(array[i]);
        }
        Log.d("hh",sb.toString());
    }
    private void printShortArray(short array[], String tag) {
        StringBuilder sb = new StringBuilder(tag);
        for(int i=0;i<array.length;i++) {
            sb.append(";").append(array[i]);
        }
        Log.d(tag,sb.toString());
    }
}
```

Ниже описаны основные элементы приведенного кода.

- RegularPolygon — ***конструктор***, который берет в качестве данных для ввода координаты центра, радиус и количество сторон.
- getAngleArrays — основной метод, отвечающий за подсчет углов у каждой оси правильного многоугольника. Причем предполагается, что одна из сторон многоугольника параллельна оси *X*.

- getXMultiplierArray и getYMultiplierArray — эти методы берут углы из getAngleArrays и проецируют их на оси *X* и *Y* для получения соответствующих координат, причем предполагается, что длина оси равна единице.
- calcArrays — этот метод использует getXMultiplierArray и getYMultiplierArray для получения каждой вершины и масштабирования их с целью совпадения с указанным радиусом и началом координат. После выполнения этого метода у нас будут верные координаты RegularPolygon, правда, в виде массивов плавающих чисел Java.
- getVertexBuffer — затем этот метод берет массивы координат, выраженных в форме плавающих чисел, и заполняет значениями NIO-буферы, которые необходимы для работы рисовальных методов OpenGL.
- getIndexBuffer — метод берет собранные вершины и упорядочивает их так, чтобы все треугольники складывались в итоговый многоугольник.

Остальные методы, предназначенные для работы с текстурами, работают схожим образом. Эти методы будут подробнее рассмотрены в следующем подразделе, где пойдет разговор о текстурах. Кроме того, мы покажем несколько функций печати, так как в целях отладки массивы приходится распечатывать.

Отображение квадрата при помощи RegularPolygon

Итак, мы рассмотрели основные «кирпичики». Теперь узнаем, как нарисовать квадрат, имея четырехсторонний RegularPolygon. В листинге 10.25 показан код SquareRenderer, упоминавшийся в листинге 10.21, — тогда мы рисовали квадрат при помощи одного из параметров меню.

Листинг 10.25. SquareRenderer

```
public class SquareRenderer extends AbstractRenderer
{
   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFVertexBuffer;

   // Необработанный нативный буфер, в котором содержатся индексы
   //обеспечивающие многократное использование точек.
   private ShortBuffer mIndexBuffer;

   private int numOfIndices = 0;

   private int sides = 4;

   public SquareRenderer(Context context)
   {
      prepareBuffers(sides);
   }
   private void prepareBuffers(int sides)
   {
```

```
        RegularPolygon t = new RegularPolygon(0,0,0,0.5f,sides);
        // RegularPolygon t = new RegularPolygon(1,1,0,1,sides);
        this.mFVertexBuffer = t.getVertexBuffer();
        this.mIndexBuffer = t.getIndexBuffer();
        this.numOfIndices = t.getNumberOfIndices();
        this.mFVertexBuffer.position(0);
        this.mIndexBuffer.position(0);
    }

    // переопределенный метод
    protected void draw(GL10 gl)
    {
        prepareBuffers(sides);
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
        gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
            GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
    }
}
```

Полагаем, что в этом коде все понятно. Мы построили его на основе AbstractRenderer (см. листинг 10.10), переопределили метод draw и использовали RegularPolygon, чтобы нарисовать квадрат. При выборе в листинге 10.21 правильного параметра вы увидите на экране эмулятора изображение, показанное на рис. 10.8.

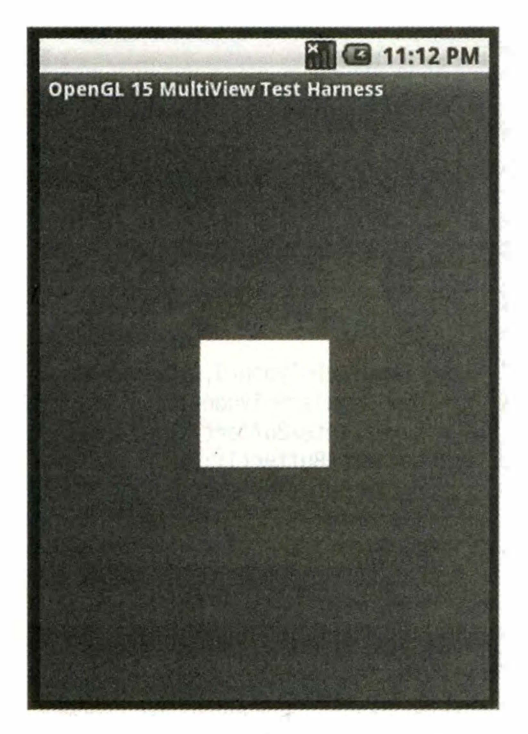


Рис. 10.8. Квадрат, нарисованный как правильный многоугольник

Анимирование RegularPolygon

Итак, мы исследовали основы рисования контуров на примере RegularPoygon. Теперь изучим пример посложнее и попробуем реализовать анимацию, в ходе которой треугольник превращается в круг. Для достижения такого эффекта нужно использовать многоугольник, количество сторон которого будет увеличиваться, скажем, раз в четыре секунды (листинг 10.26).

Листинг 10.26. PolygonRenderer

```
public class PolygonRenderer extends AbstractRenderer
{
   // Количество точек или вершин, которые мы собираемся использовать.
   private final static int VERTS = 4;

   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFVertexBuffer;

   // Необработанный нативный буфер, в котором содержатся индексы,
   // обеспечивающие многократное использование точек.
   private ShortBuffer mIndexBuffer;

   private int numOfIndices = 0;

   private long prevtime = SystemClock.uptimeMillis();

   private int sides = 3;

   public PolygonRenderer(Context context)
   {
      // EvenPolygon t = new EvenPolygon(0,0,0,1,3);
      // EvenPolygon t = new EvenPolygon(0,0,0,1,4);
      prepareBuffers(sides);
   }

   private void prepareBuffers(int sides)
   {
      RegularPolygon t = new RegularPolygon(0,0,0,1,sides);
      // RegularPolygon t = new RegularPolygon(1,1,0,1,sides);
      this.mFVertexBuffer = t.getVertexBuffer();
      this.mIndexBuffer = t.getIndexBuffer();
      this.numOfIndices = t.getNumberOfIndices();
      this.mFVertexBuffer.position(0);
      this.mIndexBuffer.position(0);
   }

   // переопределенный метод
   protected void draw(GL10 gl)
   {
      long curtime = SystemClock.uptimeMillis();
      if ((curtime - prevtime) > 2000)
```

```
      {
         prevtime = curtime;
         sides += 1;
         if (sides > 20)
         {
            sides = 3;
         }
         this.prepareBuffers(sides);
      }
      // EvenPolygon.test();
      gl.glColor4f(1.0f, 0, 0, 0.5f);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
      gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
         GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
   }
}
```

В этом коде мы просто изменяем переменную sides раз в 4 секунды. Анимация происходит тем же способом, что и в листинге 10.21 (регистрация Renderer), когда мы соотнесли PolygonRenderer с соответствующим элементом меню.

В данном случае полезно рассмотреть изменение многоугольника с течением времени. На рис. 10.9 показан шестиугольник в начале цикла.

А вот что получается к концу цикла (рис. 10.10).

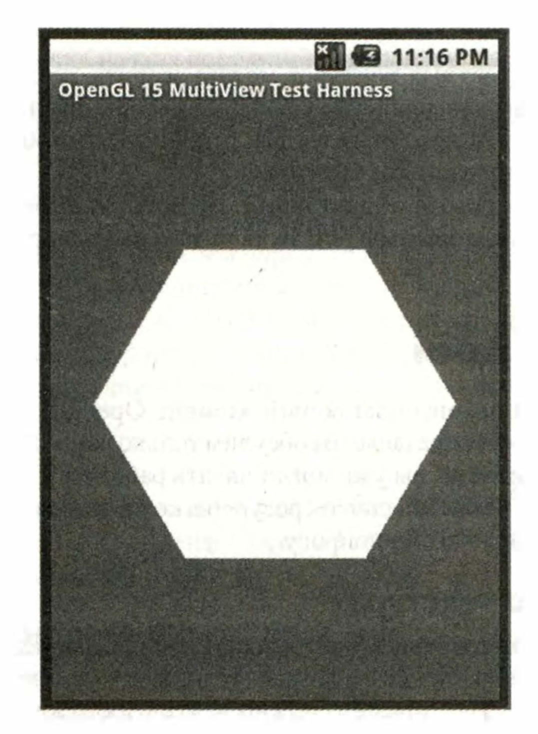


Рис. 10.9. Шестиугольник — начальный этап рисования многоугольника

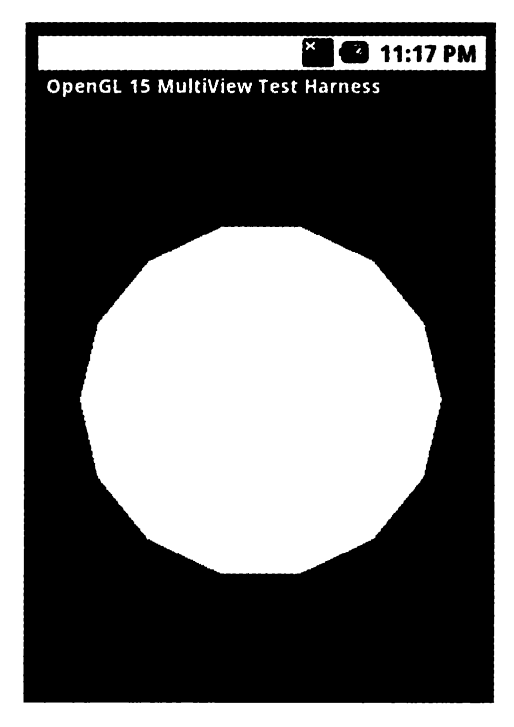


Рис. 10.10. Правильный многоугольник, по форме близкий к кругу

Данную идею абстрактных контуров можно развить, создавая более сложные контуры и даже графы сцен, включающие целые наборы других объектов, которые определяются с применением определенных типов XML и отображаются в OpenGL при помощи инстанцированных объектов.

Далее переходим к работе с текстурами. Ее можно сравнить с наклеиванием обоев на поверхности, которые мы рисовали выше (например, квадраты и многоугольники).

Работа с текстурами

Текстуры — еще один основополагающий элемент OpenGL. С ними связаны некоторые нюансы. В этом подразделе мы обсудим только наиболее фундаментальные аспекты, чтобы, прочитав ее, вы уже могли начать работать с текстурами OpenGL. В конце данной главы приведен список ресурсов, которые помогут вам приобрести более глубокие знания по этому вопросу.

Общие сведения о текстурах

Текстура OpenGL — это точечный (растровый) рисунок, накладываемый в OpenGL на определенную поверхность (в этой главе будет рассмотрено наложение текстур только на поверхности). Например, можно взять изображение почтовой марки и наложить его на квадрат, в результате он будет выглядеть как почтовая марка.

Или можно взять изображение кирпича, наложить его на прямоугольник и многократно повторить это изображение, чтобы получилась кирпичная стена.

Процесс наложения точечной текстуры на поверхность OpenGL напоминает наклеивание куска обоев (квадратной формы) на боковую поверхность объекта правильной или неправильной формы. Контуры поверхности не имеют значения, если у вас достаточно большой кусок обоев, который может накрыть ее целиком.

Однако нужно выровнять и верно сориентировать кусок обоев, чтобы изображение было правильно расположено. Для этого мы находим каждую вершину контура и точно отмечаем ее на бумаге, чтобы контуры обоев и объекта были параллельны. Если контуры поверхности сложны и у нее много вершин, то на бумаге все равно нужно обозначить каждую вершину.

Эту проблему можно рассмотреть еще в одном аспекте: представить, что вы кладете объект на поверхность лицевой стороной вверх, поверх него кладете бумагу, а потом вращаете, пока изображение, нарисованное на бумаге, не будет выровнено правильно. Теперь проткните в бумаге отверстия на каждой вершине контура. Уберите бумагу, посмотрите, где расположены отверстия, и отметьте их координаты на бумаге (предполагается, что бумага имеет разметку). Полученные координаты будут называться координатами текстуры.

Нормализованные координаты текстур

Осталось уточнить еще две детали: каков будет размер объекта и каковы размеры бумаги. В OpenGL для решения этой проблемы применяется нормализация. Предполагается, что «бумага» всегда представляет собой квадрат размером 1×1, его центр расположен в точке начала координат, а правый угол — в точке (`1,1`). Затем в OpenGL необходимо сузить поверхность вашего объекта так, чтобы он укладывался в эти границы 1×1. От программиста требуется обозначить вершины контура в квадрате, имеющем размер 1×1.

Если вспомнить, как мы рисовали наш `RegularPolygon` в листинге 10.24, то станет понятно, что в том случае мы использовали схожий метод — предполагали, что радиус круга равен единице. Затем мы определяли, где находится каждая вершина. Если предположить, что круг вписан в квадрат размером 1×1, то этот квадрат и будет нашим «куском бумаги». То есть определение координат текстуры очень напоминает определение координат вершин многоугольника. Вот почему в листинге 10.24 содержатся следующие функции для расчета координат текстур:

```
calcTextureArray()
getTextureBuffer()
```

Как вы заметили, каждая вторая функция является общей для методов `calcTextureArrays` и `calcArrays`. При изучении OpenGL важно отметить такую общность между координатами вершин и текстур.

Работа с абстрагированием общих текстур

После того как вы усвоите указанное соотношение между координатами вершин и текстур и определите координаты текстурной карты, основная часть работы будет выполнена. Далее нужно будет загрузить растровую текстуру в память и присвоить

текстуре ID, чтобы ее можно было многократно использовать. Затем, чтобы можно было одновременно загружать несколько текстур, применяется механизм задания текущей текстуры через ее ID. При работе графического конвейера координаты текстуры указываются вместе с чертежными координатами. Затем вы рисуете.

Поскольку загружать текстуры приходится довольно часто, мы выделили этот процесс, чтобы создать абстрактный класс, называемый SingleAbstractTextureRendererи наследующий свойства AbstractRenderer.

В листинге 10.27 приведен весь исходный код, необходимый для установки текстур.

Листинг 10.27. Обеспечение абстрагирования наложения одиночных текстур

```
public abstract class AbstractSingleTexturedRenderer
extends AbstractRenderer
{
   int mTextureID;
   int mImageResourceId;
   Context mContext;
   public AbstractSingleTexturedRenderer(Context ctx,
                     int imageResourceId) {
      mImageResourceId = imageResourceId;
      mContext = ctx;
   }

   public void onSurfaceCreated(GL10 gl, EGLConfig eglConfig) {
      super.onSurfaceCreated(gl, eglConfig);
      gl.glEnable(GL10.GL_TEXTURE_2D);
      prepareTexture(gl);
   }
   private void prepareTexture(GL10 gl)
   {
      int[] textures = new int[1];
      gl.glGenTextures(1, textures, 0);

      mTextureID = textures[0];
      gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);

      gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_MIN_FILTER,
         GL10.GL_NEAREST);

      gl.glTexParameterf(GL10.GL_TEXTURE_2D,
         GL10.GL_TEXTURE_MAG_FILTER,
         GL10.GL_LINEAR);

      gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
         GL10.GL_CLAMP_TO_EDGE);
      gl.glTexParameterf(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
         GL10.GL_CLAMP_TO_EDGE);

      gl.glTexEnvf(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
```

```
            GL10.GL_REPLACE);

        InputStream is = mContext.getResources()
            .openRawResource(this.mImageResourceId);
        Bitmap bitmap;
        try {
            bitmap = BitmapFactory.decodeStream(is);
        } finally {
            try {
                is.close();
            } catch(IOException e) {
                // игнорировать
            }
        }

        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
        bitmap.recycle();
    }

    public void onDrawFrame(GL10 gl)
    {
        gl.glDisable(GL10.GL_DITHER);
        gl.glTexEnvx(GL10.GL_TEXTURE_ENV, GL10.GL_TEXTURE_ENV_MODE,
            GL10.GL_MODULATE);

        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        GLU.gluLookAt(gl, 0, 0, -5, 0f, 0f, 0f, 0f, 1.0f, 0.0f);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);

        gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);

        gl.glActiveTexture(GL10.GL_TEXTURE0);
        gl.glBindTexture(GL10.GL_TEXTURE_2D, mTextureID);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_S,
            GL10.GL_REPEAT);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D, GL10.GL_TEXTURE_WRAP_T,
            GL10.GL_REPEAT);

        draw(gl);
    }
}
```

В этом коде загружается одиночная текстура (точечный рисунок), которая подготавливается в методе onSurfaceCreated. Код onDrawFrame, как и AbstractRenderer, устанавливает параметры вашей области рисования (drawing space) — так, чтобы координаты были применимы. В зависимости от ситуации, этот код нужно откорректировать, чтобы задать оптимальную зону видимости.

Обратите также внимание на то, что конструктор берет точечную текстуру и сохраняет для последующего использования. В зависимости от того, с каким количеством текстур вы работаете, абстрактные классы нужно соответствующим образом изменить.

В листинге 10.27 встречаются следующие API, обычные при работе с текстурами.

- glGenTextures — при помощи этого метода OpenGL генерируются уникальные ID для текстур, чтобы на текстуры в дальнейшем можно было ссылаться посредством данных ID. После того как точечная текстура будет загружена при помощи GLUtils.texImage2D, она прикрепляется к определенному ID. Пока текстура прикреплена к ID, сгенерированному посредством glGenTextures, он остается обычным идентификатором. В литературе по OpenGL такие идентификаторы (натуральные числа) называются именами текстур.
- glBindTexture — при помощи этого метода OpenGL текстура, загруженная в настоящий момент, прикрепляется к ID, полученному методом glGenTextures.
- glTexParameter — существует очень много параметров, которые можно устанавливать при работе с текстурами. В этом API определяются такие параметры. В качестве примера можно привести GL_REPEAT, GL_CLAMP и др. Например, GL_REPEAT позволяет многократно копировать точечный рисунок, если он гораздо меньше объекта, на который накладывается текстура. Подробный список параметров приводится на сайте Khronos по OpenGL ES: **http://www.khronos.org/opengles/documentation/opengles1_0/html/glTexParameter.html**.
- glTexEnv — еще некоторые параметры, касающиеся текстур, указываются при помощи метода glTexEnv. Примеры значений — GL_DECAL, GL_MODULATE, GL_BLEND, GL_REPLACE и т. д. Например, при использовании GL_DECAL текстура покрывает лежащий ниже объект. GL_MODULATE, как следует из названия, изменяет цвета располагающегося ниже объекта, но не заменяет их. По следующей ссылке приведен полный список параметров для этого API: **http://www.khronos.org/opengles/documentation/opengles1_0/html/glTexEnv.html**.
- GLUtils.texImage2D — это API Android, позволяющий загрузить точечный рисунок и использовать его для текстурирования. Внутри системы этот API вызывает glTexImage2D OpenGL.
- glActiveTexture — задает текстуру с указанным ID в качестве активной.
- glTexCoordpointer — этот метод OpenGL используется для указания координат текстуры. Все координаты должны совпадать с указанными в glVertexPointer.

О большинстве из этих API можно прочитать в справке по OpenGL ES по следующей ссылке: **http://www.khronos.org/opengles/documentation/opengles1_0/html/index.html**.

Рисование с применением текстур

Когда точечный рисунок будет загружен и задан в качестве текстуры, мы сможем задействовать RegularPolygon и использовать координаты текстур и вершин для рисования правильного многоугольника вместе с текстурой. В листинге 10.28 представлен класс, при помощи которого рисуется текстурированный квадрат.

Листинг 10.28. TexturedSquareRenderer

```
public class TexturedSquareRenderer extends AbstractSingleTexturedRenderer
{
   // Количество точек или вершин, которые мы собираемся использовать.
   private final static int VERTS = 4;

   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFVertexBuffer;

   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFTextureBuffer;

   // Необработанный нативный буфер, в котором содержатся индексы
   //обеспечивающие многократное использование точек.
   private ShortBuffer mIndexBuffer;

   private int numOfIndices = 0;

   private int sides = 4;

   public TexturedSquareRenderer(Context context)
   {
      super(context,com.ai.android.OpenGL.R.drawable.robot);
      prepareBuffers(sides);
   }

   private void prepareBuffers(int sides)
   {
      RegularPolygon t = new RegularPolygon(0,0,0,0.5f,sides);
      this.mFVertexBuffer = t.getVertexBuffer();
      this.mFTextureBuffer = t.getTextureBuffer();
      this.mIndexBuffer = t.getIndexBuffer();
      this.numOfIndices = t.getNumberOfIndices();
      this.mFVertexBuffer.position(0);
      this.mIndexBuffer.position(0);
      this.mFTextureBuffer.position(0);
   }

   // переопределенный метод
   protected void draw(GL10 gl)
   {
      prepareBuffers(sides);
      gl.glEnable(GL10.GL_TEXTURE_2D);
      gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
      gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mFTextureBuffer);
      gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
         GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
   }
}
```

Как видите, большая часть работы выполняется абстрактным классом, предназначенным для отображения текстур, и RegularPolygon (расчет совпадения текстуры с вершинами, см. листинг 10.24). При выборе меню в листинге 10.21 отобразится квадрат с текстурой, показанный на рис. 10.11.

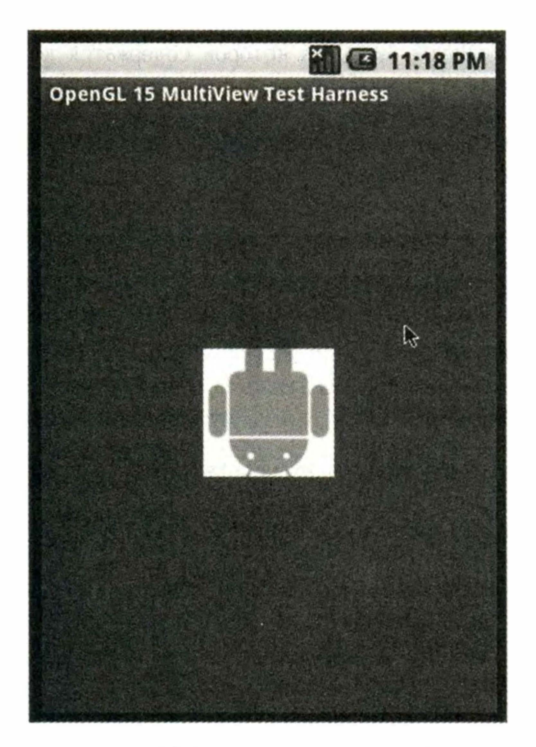


Рис. 10.11. Квадрат с текстурой

Рисование нескольких фигур

До сих пор во всех примерах этой главы мы рисовали одну простую фигуру по стандартному образцу. Образец таков: задаются вершины, загружается текстура, задаются координаты текстуры, рисуется одиночная фигура. А что, если нужно нарисовать две фигуры? Что делать, если требуется нарисовать треугольник обычным способом — с указанием вершин, а затем многоугольник с применением контуров? Как указать общие вершины? Нужно ли один раз указать вершины для обоих объектов, а потом вызвать метод для рисования? Эти важные вопросы мы рассмотрим в заключительном подразделе главы.

Оказывается, между любыми двумя вызовами draw( ), относящимися к интерфейсу рендеринга Android OpenGL, можно применить несколько методов glDraw. Между вызовами glDraw можно задавать новые вершины и текстуры. Затем результат выполнения всех этих методов отобразится на экране — после завершения метода draw( ).

Есть еще один полезный прием, который пригодится при рисовании нескольких фигур в OpenGL. Вспомните многоугольники, которые мы рисовали выше. Такие многоугольники способны воспроизводить сами себя в любой точке, в качестве данных ввода (и начала координат) берется эта точка. Практика показывает, что OpenGL может выполнять такие операции нативным способом. При этом вы всегда устанавливаете `RegularPolygon` в точке `(0,0,0)`, а механизм преобразования, имеющийся в OpenGL, перемещает фигуру с точки начала координат на желаемую позицию. Вы можете проделать то же самое и с другим многоугольником. В результате на экране окажется два многоугольника с разными точками начала координат.

В листинге 10.29 показано, как многократно нарисовать текстурированный многоугольник.

Листинг 10.29. Рендеринг текстурированного многоугольника

```
public class TexturedPolygonRenderer extends AbstractSingleTexturedRenderer
{
   // Количество точек или вершин, которые мы собираемся использовать.
   private final static int VERTS = 4;

   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFVertexBuffer;

   // Необработанный нативный буфер, в котором содержатся координаты точек.
   private FloatBuffer mFTextureBuffer;

   // Необработанный нативный буфер, в котором содержатся индексы,
   // обеспечивающие многократное использование точек.
   private ShortBuffer mIndexBuffer;

   private int numOfIndices = 0;

   private long prevtime = SystemClock.uptimeMillis();
   private int sides = 3;

   public TexturedPolygonRenderer(Context context)
   {
      super(context,com.ai.android.OpenGL.R.drawable.robot);
      // EvenPolygon t = new EvenPolygon(0,0,0,1,3);
      // EvenPolygon t = new EvenPolygon(0,0,0,1,4);
      prepareBuffers(sides);
   }

   private void prepareBuffers(int sides)
   {
   RegularPolygon t = new RegularPolygon(0,0,0,0.5f,sides);
   // RegularPolygon t = new RegularPolygon(1,1,0,1,sides);
   this.mFVertexBuffer = t.getVertexBuffer();
   this.mFTextureBuffer = t.getTextureBuffer();
```

```
    this.mIndexBuffer = t.getIndexBuffer();
    this.numOfIndices = t.getNumberOfIndices();
    this.mFVertexBuffer.position(0);
    this.mIndexBuffer.position(0);
    this.mFTextureBuffer.position(0);
    }

    // переопределенный метод
    protected void draw(GL10 gl)
    {
        long curtime = SystemClock.uptimeMillis();
        if ((curtime - prevtime) > 2000)
        {
            prevtime = curtime;
            sides += 1;
            if (sides > 20)
            {
                sides = 3;
            }
            this.prepareBuffers(sides);
        }
        gl.glEnable(GL10.GL_TEXTURE_2D);

        // сначала рисуем слева
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, mFVertexBuffer);
        gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, mFTextureBuffer);

        gl.glPushMatrix();
        gl.glScalef(0.5f, 0.5f, 1.0f);
        gl.glTranslatef(0.5f,0, 0);
        gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
            GL10.GL_UNSIGNED_SHORT, mIndexBuffer);

        // потом рисуем справа
        gl.glPopMatrix();
        gl.glPushMatrix();
        gl.glScalef(0.5f, 0.5f, 1.0f);
        gl.glTranslatef(-0.5f,0, 0);
        gl.glDrawElements(GL10.GL_TRIANGLES, this.numOfIndices,
            GL10.GL_UNSIGNED_SHORT, mIndexBuffer);
        gl.glPopMatrix();
    }
}
```

В этом примере в обобщенном виде показаны несколько концепций:

- рисование с применением контуров;
- рисование нескольких контуров с использованием матриц преобразований;
- применение текстур;
- использование анимации.

Основная часть кода, отвечающая в листинге 10.29 за многократное рисование, — это метод `draw()`. Мы выделили соответствующие строки этого метода. Как видите, при активации `draw()` мы дважды вызывали `glDrawElements`. Каждый раз мы применяли при рисовании простые формы, независимо от рисования второго экземпляра.

Еще один момент, нуждающийся в разъяснении, — это использование матриц преобразований (transformation matrices). Всякий раз при вызове метода `glDrawElements()` применяется отдельная матрица преобразований. Если бы нам потребовалось изменить такую матрицу, чтобы изменить положение рисунка (или любую другую его характеристику), то для правильного рисования следующего элемента ее нужно было бы вернуть в исходное состояние. Для этого в OpenGL с матрицами используются операции push and pop.

На рис. 10.12 показан результат выполнения этого кода (снимок сделан перед началом анимации).



Рис. 10.12. Два текстурированных многоугольника

А вот что мы видим примерно в середине анимации (рис. 10.13).

На этом мы завершаем изучение еще одного важного аспекта OpenGL. В данном разделе было показано, как собрать вместе различные текстуры или сцены и нарисовать их в паре так, чтобы в результате получилась довольно сложная сцена OpenGL.

В следующем разделе мы перечислим важные ресурсы, которые могут дать вам пищу для размышлений и помочь при дальнейшем изучении OpenGL.

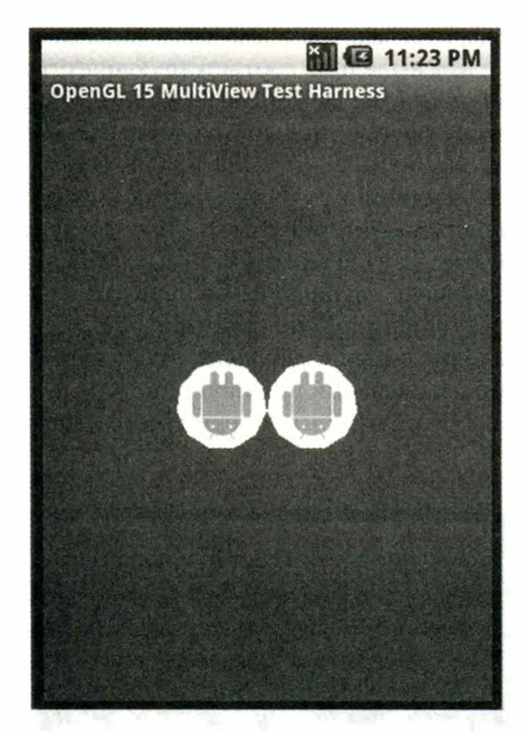


Рис. 10.13. Два текстурированных круга

Ресурсы по OpenGL

Авторы считают, что следующие сайты будут полезны при изучении OpenGL и работе в ней.

- Ссылка на пакет Android `android.opengl`: http://developer.android.com/reference/android/opengl/GLSurfaceView.html.
- Справочный мануал Khronos Group, посвященный OpenGL ES: http://www.khronos.org/opengles/documentation/opengles1_0/html/index.html.
- Руководство по программированию в OpenGL («Красная книга»): http://www.glprogramming.com/red/.
- Очень хорошая статья о наложении текстур от Microsoft: http://msdn.microsoft.com/en-us/library/ms970772(printer).aspx.
- Очень глубокий курс лекций по OpenGL авторства Уэйна О. Кохрана из Вашингтонского государственного университета: http://ezekiel.vancouver.wsu.edu/~cs442/.
- Документация по JSR 239, привязке Java к OpenGL ES API: http://java.sun.com/javame/reference/apis/jsr239/.
- Исследование одного из авторов этой книги, посвященное OpenGL, вы можете прочитать по адресу http://www.satyakomatineni.com/akc/display?url=NotesIMPTitlesURL&ownerUserId=satya&folderName=OpenGL&order_by_format=news.

- Исследование одного из авторов этой книги, посвященное текстурам в OpenGL, вы можете прочитать на этой странице: http://www.satyakomatineni.com/akc/display?url=DisplayNoteIMPURL&reportId=3190&ownerUserId=satya.

Резюме

Мы подробно изучили основополагающие моменты, связанные с OpenGL. Эта информация будет вам особенно полезна, если раньше вы не программировали в OpenGL. Мы надеемся, что эта большая глава станет для вас вводным курсом OpenGL и пригодится при работе не только в Android, но и в других системах.

В данной главе мы изучили основы OprnGL; рассмотрели специфичные для Android API, обеспечивающие работу со стандартными API OpenGL; поговорили о контурах и текстурах, узнали, как пользоваться графическим конвейером для рисования нескольких фигур сразу.

Желаем вам и дальше совершенствовать свои навыки работы в OpenGL, в том числе при помощи ресурсов, перечисленных в последнем разделе главы. По мере усложнения процессоров мобильных устройств OpenGL для мобильных платформ должна обогатиться в нескольких следующих версиях большим количеством новых разработок.

11 Управление настройками и их организация

Как и многие другие SDK, Android поддерживает пользовательские настройки (preferences). Android отслеживает настройки, касающиеся пользователей программы, а также самой программы. Например, пользователь Microsoft Outlook может задать настройку, согласно которой электронные сообщения будут просматриваться определенным образом, и в самом Microsoft Outlook по умолчанию будут действовать параметры, которые пользователь сможет изменять. Но хотя теоретически Android отслеживает настройки, касающиеся как самого приложения, так и конкретного пользователя, он не отличает первых от вторых. Причина этого заключается в том, что программы Android обычно работают на устройстве, предназначенном только для одного пользователя (ведь не принято давать кому-нибудь свой сотовый телефон). Итак, в Android настройки воспринимаются в первую очередь как *настройки приложения* (application preferences). Под ними понимаются и пользовательские настройки, и заданные по умолчанию настройки программы.

Когда вы познакомитесь с поддержкой настроек в Android, вы, несомненно, будете впечатлены. В Android для работы с настройками предлагается надежный и гибкий фреймворк. В системе есть простые API, скрывающие процессы считывания и сохранения настроек, а также встроенные пользовательские интерфейсы, при помощи которых пользователь может задавать настройки. Все эти функции будут рассмотрены в следующих разделах.

Исследование фреймворка настроек

Прежде чем подробно заняться изучением настроек в Android, спланируем сценарий, согласно которому нам потребуется использовать настройки, и научимся с ними обращаться. Предположим, мы пишем программу, предназначенную для поиска авиарейсов. По умолчанию программа отображает самые дешевые рейсы, но пользователь может выбрать сортировку по наименьшему количеству остановок или по отдельно взятому маршруту. Как выполнить такую задачу?

ListPreference

Очевидно, нам потребуется написать интерфейс (UI), в котором пользователь может просматривать и сортировать параметры. В списке будут содержаться пере-

ключатели для каждой настройки, причем один из параметров (заданный по умолчанию) будет выбран сразу. Для решения этой задачи во фреймворке настроек Android требуется выполнить минимум работы. Сначала нужно создать XML-файл с настройками, а затем применить встроенный класс явления, предназначенный для показа и сохранения настроек. В листинге 11.1 этот процесс показан подробно.

Листинг 11.1. XML-файл с настройками программы выбора авиарейсов и соответствующий класс явления

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/xml/flightoptions.xml -->
<PreferenceScreen
        xmlns:android="http://schemas.android.com/apk/res/android"
      android:key="flight_option_preference"
      android:title="@string/prefTitle"
      android:summary="@string/prefSummary">

   <ListPreference
      android:key="@string/selected_flight_sort_option"
      android:title="@string/listTitle"
      android:summary="@string/listSummary"
      android:entries="@array/flight_sort_options"
      android:entryValues="@array/flight_sort_options_values"
      android:dialogTitle="@string/dialogTitle"
      android:defaultValue="@string/flight_sort_option_default_value" />

</PreferenceScreen>

package com.syh;

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class FlightPreferenceActivity extends PreferenceActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.flightoptions);
   }
}
```

В листинге 11.1 приведен фрагмент XML-файла, в котором показана установка настроек выбора авиарейсов. Кроме того, в листинге содержится класс явления, загружающий XML-файл настроек. Рассмотрим сначала XML. Фреймворк настроек в Android является комплексным (end-to-end). Это значит, что в нем можно задавать настройки, отображать параметры пользователю и сохранять настройки, выбранные пользователем, в хранилище данных. Настройки определяются в XML по адресу /res/xml/. Чтобы показать настройки пользователю, нужно написать класс явления, дополняющий стандартный класс Android android.preference.PreferenceActivity, а затем применить метод addPreferencesFromResource(), чтобы

добавить ресурс в коллекцию ресурсов явления. Остальную работу (отображение и сохранение) выполняет фреймворк.

В нашем сценарии с выбором авиарейсов мы создали файл `flightoptions.xml` по адресу `/res/xml/flightoptions.xml`. Затем создается класс явления, который называется `FlightPreferenceActivity` и дополняет класс `android.preference.PreferenceActivity`. Потом мы вызываем метод `addPreferencesFromResource()`, передавая с ним `R.xml.flightoptions`. Обратите внимание — XML-файл ресурсов, описывающий настройки, указывает на несколько строковых ресурсов. Чтобы компиляция прошла успешно, в проект нужно добавить строковые ресурсы. Скоро мы покажем, как это делается. Пока давайте рассмотрим пользовательский интерфейс, который мы создали в листинге 11.1 (рис. 11.1).

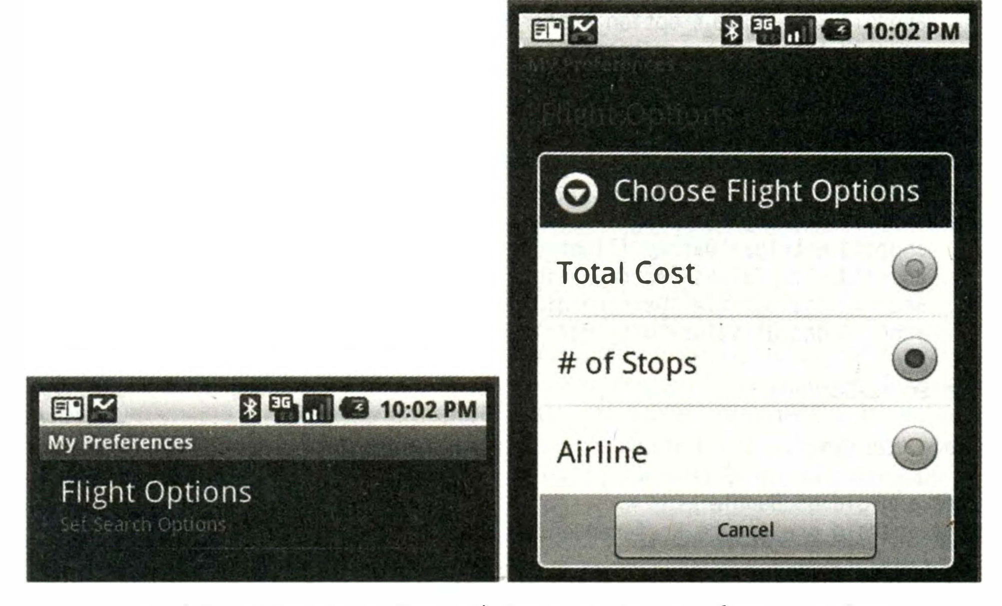


Рис. 11.1. Пользовательский интерфейс с настройками выбора авиарейсов

На рис. 11.1 показаны два вида. Вид, расположенный слева, называется *экраном настроек* (preference screen), а размещенный справа — *списком настроек* (list preference). Когда пользователь выбирает **Flight Options** (Параметры рейса), на экран в виде модального диалогового окна выводится вид **Choose Flight Options** (Выбор параметров рейса), причем каждому из параметров соответствует положение переключателя. При выборе настройки она автоматически сохраняется, и вид закрывается. При следующем открытии экрана параметров в окне отображается последний сохраненный выбор.

Итак, XML-файл настроек и связанный с ним класс явления показаны в листинге 11.1. В этом коде определяется `PreferenceScreen`, а затем создается его дочерний элемент — `ListPreference`. Для `PreferenceScreen` устанавливаются три свойства: `key`, `title` и `summary`. `key` — это строка, которую можно использовать для ссылки на элемент программными методами (похоже на `android:id`), `title` — это заголовок экрана (**Flight Options**), а `summary` — это описание назначения экрана, ото-

бражается под заголовком более мелким шрифтом (в данном случае — **Set Search Options**). Для списка настроек задаются свойства key, title и summary, а также атрибуты для entries, entryValues, dialogTitle и defaultValue. Описание этих атрибутов приведено в табл. 11.1.

Таблица 11.1. Некоторые атрибуты android.preference.ListPreference

Атрибут	Описание
android:key	Название или ключ параметра (например, selected_flight_sort_option)
android:title	Заголовок параметра
android:summary	Краткое описание параметра
android:entries	Текст элементов списка, которые выбираются в качестве параметров
android:entryValues	Определяет ключ или значение для каждого элемента. Обратите внимание: с каждым элементом связаны определенный текст и значение. Текст определяется в entries, а значения — в entryValues
android:dialogTitle	Заголовок диалогового окна; используется, если вид отображается как модальное диалоговое окно
android:defaultValue	Параметр, по умолчанию выбираемый из списка

Доработайте наш пример, добавив нужные файлы (или измените имеющиеся) так, как это описано в листинге 11.2.

Листинг 11.2. Доработка проекта

```
<?xml version="1.0" encoding="utf-8"?>
<!-- это файл /res/values/arrays.xml -->
<resources>
<string-array name="flight_sort_options">
   <item>Total Cost</item>
   <item># of Stops</item>
   <item>Airline</item>
</string-array>
<string-array name="flight_sort_options_values">
   <item>0</item>
   <item>1</item>
   <item>2</item>
</string-array>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<!-- это файл /res/values/strings.xml -->
<resources>
   <string name="app_name">Preferences Demo</string>
   <string name="prefTitle">My Preferences</string>
   <string name="prefSummary">Set Flight Option Preferences</string>
   <string name="flight_sort_option_default_value">1</string>
   <string name="dialogTitle">Choose Flight Options</string>
   <string name="listSummary">Set Search Options</string>
   <string name="listTitle">Flight Options</string>
   <string name=
      "selected_flight_sort_option">selected_flight_sort_option</string>
   <string name="menu_prefs_title">Settings</string>
```

```
   <string name="menu_quit_title">Quit</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/menu/mainmenu.xml -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/menu_prefs"
   android:title="@string/menu_prefs_title"
   />
<item android:id="@+id/menu_quit"
   android:title="@string/menu_quit_title"
   />
</menu>

<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent
   >

<TextView android:text="" android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   />

</LinearLayout>

// это файл MainActivity.java
import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.TextView;

public class MainActivity extends Activity {
   private TextView tv = null;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      tv = (TextView)findViewById(R.id.text1);

      setOptionText();
```

```
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.mainmenu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected (MenuItem item)
    {
        if (item.getItemId() == R.id.menu_prefs)
        {
            Intent intent = new Intent()
                    .setClass(this, com.syh.FlightPreferenceActivity.class);
            this.startActivityForResult(intent, 0);
        }
        else if (item.getItemId() == R.id.menu_quit)
        {
            finish();
        }
        return true;
    }

    @Override
    public void onActivityResult(int reqCode, int resCode, Intent data)
    {
        super.onActivityResult(reqCode, resCode, data);
        setOptionText();
    }

    private void setOptionText()
    {
        SharedPreferences prefs =
                getSharedPreferences("com.syh_preferences", 0);
        String option = prefs.getString(this.getResources().getString(
                R.string.selected_flight_sort_option),
this.getResources().getString(R.string.flight_sort_option_default_value));
        String[] optionText =
this.getResources().getStringArray(R.array.flight_sort_options);

        tv.setText("option value is " + option + " (" +
                optionText[Integer.parseInt(option)] + ")");
    }
}

<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл AndroidManifest.xml -->
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.syh"
      android:versionCode="1"
      android:versionName="1.0">
   <application android:icon="@drawable/icon"
            android:label="@string/app_name">
      <activity android:name=".MainActivity"
            android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

      <activity android:name=".FlightPreferenceActivity"
            android:label="@string/prefTitle">
         <intent-filter>
            <action android:name=
               "com.syh.intent.action.FlightPreferences" />
            <category android:name=
               "android.intent.category.PREFERENCE" />
         </intent-filter>
      </activity>

   </application>
   <uses-sdk android:minSdkVersion="3" />

</manifest>
```

После внесения этих изменений и запуска приложения вы сначала увидите простое текстовое сообщение: **option value is 1 (# of Stops)** (значение параметра равно 1 (количество остановок)). Нажмите кнопку **Menu** (Меню), а затем **Settings** (Настройки), чтобы перейти к `PreferenceActivity`. Когда сделаете необходимые настройки, нажмите стрелку «назад» — внесенные изменения сразу применятся к тексту параметра.

Сначала мы добавили файл `/res/values/arrays.xml`. В нем содержится два строковых массива, в которых мы реализуем варианты выбора параметров. В первом массиве находится текст, который будет отображаться на экране, а во втором — значения, которые мы будем получать в ответ при вызове методов, плюс значения, которые будут сохраняться в XML-файле настроек. Мы будем использовать в программе значения индексов элементов массива 0, 1 и 2 для `flight_sort_options_values`. Выбираем любое значение и с его помощью будем запускать программу. Если параметр должен иметь именно числовое значение (например, параметр представляет собой начальное значение таймера обратного отсчета), то можно использовать 60, 120, 300 и т. д. Эти значения не обязательно должны быть числовыми, главное, чтобы они были понятны разработчику. Пользователь не увидит этих значений, если вы специально их не отобразите. На экране будет выводиться только текст из строкового массива `flight_sort_options`.

Как мы сказали выше, во фреймворке Android также предусмотрена функция сохранения настроек. Например, если пользователь выбирает параметр сортировки, Android сохраняет сделанный выбор в XML-файле в каталоге /data конкретной программы (рис. 11.2).

com.google.android.googleapps	2008-09-27	03:37	drwxr-xr-x
com.google.android.street	2008-09-27	03:37	drwxr-xr-x
com.my.client	2008-12-22	19:51	drwxr-xr-x
com.my.services	2008-12-22	19:36	drwxr-xr-x
com.sayed	2008-12-15	02:16	drwxr-xr-x
com.sayedhashimi	2008-12-03	02:22	drwxr-xr-x
com.syh	2008-12-12	02:30	drwxr-xr-x
lib	2008-12-12	02:30	drwxr-xr-x
shared_prefs	2009-01-18	01:26	drwxrwx--x

Рис. 11.2. Путь к сохраненным настройкам программы

Точный путь к файлу — /data/data/[PACKAGE_NAME]/shared_prefs/[PACKAGE_NAME]_preferences.xml. В листинге 11.3 показан файл com.syh_preferences.xml из нашего примера.

Листинг 11.3. Сохраненные настройки из нашего примера

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
   <string name="selected_flight_sort_option">1</string>
</map>
```

Как видите, фреймворк настроек сохраняет значение, выбранное для элемента из списка настроек, при помощи спискового атрибута key. Обратите также внимание, что для элемента сохраняется именно *значение*, а не текст. Здесь нужно быть внимательным, поскольку в XML-файле настроек сохраняется только значение, но не текст. При каждом обновлении программы, изменении текста параметров либо добавлении элементов в строковые массивы все значения, сохраненные в XML-файле настроек, нужно специально обновлять, внося соответствующие изменения. При обновлении приложения XML-файл настроек сохраняется. Если в файле настроек указано значение 1 и оно соответствует «количеству остановок», то после обновления оно также должно соответствовать «количеству остановок».

Дальше в нашем примере идет файл /res/values/strings.xml. Мы добавили несколько строк в качестве заголовков, кратких описаний и названий для элементов меню. Две строки требуют особого внимания. Первая — flight_sort_option_default_value. В качестве стандартного значения (default value) мы указали 1, в нашем примере это значение соответствует «количеству остановок». Обычно следует выбирать стандартное значение для каждого параметра. Если стандартное значение не было задано, а другое значение еще не выбрано, то методы, возвращающие значение параметра, будут возвращать null. В таком случае в коде потребуется обрабатывать нулевые значения. Еще одна интересная строка — selected_flight_sort_option. Строго говоря, пользователь ее не увидит. Поэтому нам не придется записывать ее в файл strings.xml, контент которого будет переводиться при локализации

программы на других языках. Но поскольку это строковое значение является ключом, используемым при вызове метода для нахождения значения, то, создавая ID вне метода, во время компиляции мы можем гарантировать, что не допустим опечатки в названии ключа.

Третий добавленный нами файл — /res/menu/mainmenu.xml. Предполагается, что вам понадобится доступ к виду настроек через меню, а не при помощи кнопки. Этот файл представляет собой меню нашего приложения.

Четвертый файл — /res/layout/main.xml. Это основной пользовательский интерфейс нашего приложения. Итак, мы обсудили использование настроек с применением специального класса явления PreferenceActivity. Но мы собирались применять настройки в основном явлении, а не в PreferenceActivity. Следовательно, нам нужен способ для получения настроек из другого явления. В данном случае шаблон представляет собой обычный TextView, в котором отображается актуальное значение параметра из программы для выбора авиарейсов.

Далее следует исходный код явления MainActivity. Это основное явление, при помощи которого происходит управление TextView. Затем вызывается метод для считывания актуального значения параметра, которое затем помешается в TextView. Мы настраиваем меню и обратный вызов меню. В рамках обратного вызова меню мы запускаем намерение (Intent) для FlightPreferenceActivity. Когда Intent настроек возвращается, мы вызываем метод setOptionText() для обновления TextView.

Самое интересное — это метод setOptionText(). При работе с ним мы сначала получаем доступ к управлению настройками, ставя ссылку на название соответствующего XML-файла настроек. Разумеется, необходимо знать, какое имя файла будет использовать образец, описанный выше. Второй параметр касается того, собираемся ли мы только считывать значения либо еще и записывать их. Этот вопрос мы подробно рассмотрим чуть ниже. Поставив ссылку на настройки, нужно вызвать соответствующие методы для нахождения значений. В нашем случае вызываем getString(), так как знаем, что значения настроек представляют собой строки. Первый аргумент — это строковое значение ключа параметра. Ранее мы отмечали, что нужно использовать ID, чтобы застраховаться при написании программы от опечаток. Кроме того, мы можем просто применять в качестве первого аргумента строку selected_flight_sort_option. Этот способ удобен, так как наша задача — писать максимально компактные и быстрые программы. В качестве второго аргумента указывается стандартное значение на случай, если искомого значения в файле XML не окажется. Если программа запускается впервые, у вас нет XML-файла с настройками, поэтому, не указав второй аргумент, при первом запуске программы вы неизбежно получите null. Это произойдет, даже если указать стандартное значение параметра в спецификации ListPreference в flightoptions.xml. В нашем примере мы задали стандартное значение, и для его считывания можно применить код метода setOptionText(). Обратите внимание — если бы мы не использовали ID со стандартным значением, было бы гораздо тяжелее считать его прямо из ListPreference. Мы отображаем не только значение настройки, но и соответствующий текст. В нашем примере мы использовали сокращенный вариант, так как применяли индексы массива для значений flight_sort_options_values. Просто преобразовав значение в int, мы знаем, какую строку нужно считывать из flight_sort_options. Если бы мы использовали в flight_sort_options_values другой набор параметров, нам понадобилось бы определить индекс элемента, являющегося нашей настройкой, затем

вернуться и при помощи этого индекса получить текст из настройки, находящейся в flight_sort_options.

Последний файл в приведенном примере — AndroidManifest.xml. Поскольку сейчас в приложении имеется два явления, нужны два тега <activity>. Первое явление — стандартное, относится к категории LAUNCHER. Второе — PreferenceActivity, то есть мы выбираем для действия название, руководствуясь при этом правилами именования намерений, и устанавливаем категорию PREFERENCE. Скорее всего, не потребуется, чтобы PreferenceActivity отображалось со всеми другими нашими приложениями, поэтому мы и не используем с ним вариант LAUNCHER.

Ссылку на настройки немного проще получить из явления, дополняющего PreferenceActivity. Мы не будем вызывать getPreferences(), а сделаем так:

```
SharedPreferences prefs = getPreferenceManager().getDefaultSharedPreferences(this)
;
```

Управление настройками при помощи программирования

Излишне говорить о том, что может возникнуть необходимость доступа к элементам управления настройками с помощью программных методов. Например, что делать, если понадобится предоставить entries и entryValues для ListPreference во время исполнения программы? Можно определить элементы управления настройками подобно тому, как задаются элементы управления в файлах шаблонов и явлениях, а потом получать доступ к этим элементам. Например, для доступа к списку настроек, определенному в листинге 11.1, нужно вызвать метод findPreference() явления PreferenceActivity, передав ключ (key) настройки (обратите внимание на сходство с findViewById()). После этого элемент управления помещается в ListPreference и с ним можно работать. Например, если вы хотите установить записи ListPreference, вызовите метод setEntries() и т. д.

Кроме того, при помощи кода можно создавать настройки или выполнять с ними другие операции. Подробнее мы поговорим об этом в главе 13.

Теперь вы представляете, как в Android построена работа с настройками. Мы знаете, что в Android есть встроенные пользовательские интерфейсы для показа настроек и их сохранения. К тому же в Android есть класс android.preference.PreferenceActivity, дополняемый при внедрении настроек в программу. В этом классе содержатся API, предназначенные для загрузки настроек, прикрепления их к фреймворку и его расширения.

Мы рассмотрели, как обращаться с видом ListPreference. Теперь научимся работать с другими элементами фреймворка настроек Android. Речь пойдет о видах CheckBoxPreference, EditTextPreference и RingtonePreference.

CheckBoxPreference

Как вы уже видели, в настройке ListPreference список отображается как элемент пользовательского интерфейса. Схожим образом в настройке CheckBoxPreference показывается виджет-флажок, представляющий собой элемент пользовательского интерфейса.

Развивая наш пример с программой для поиска авиарейсов, предположим, что мы предоставляем пользователю возможность создавать список колонок, в которых он хочет видеть результат поиска. Эта настройка отображает все доступные колонки и позволяет пользователю отбирать из них нужные, проставляя соответствующие флажки. Пользовательский интерфейс для данного примера показан на рис. 11.3, а XML-файл с настройками — в листинге 11.4.

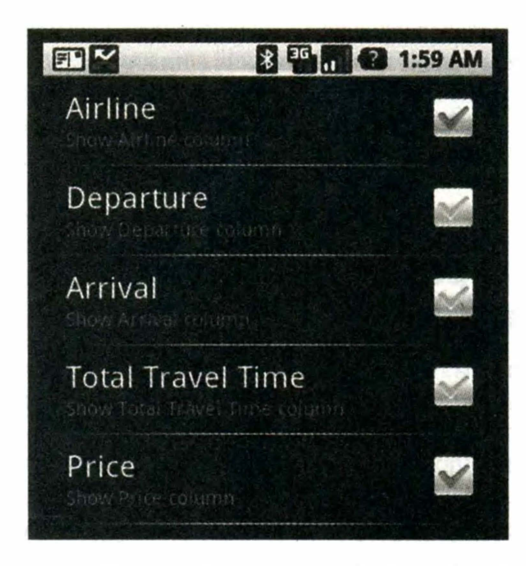


Рис. 11.3. Пользовательский интерфейс для выбора настроек при помощи флажков

Листинг 11.4. Использование CheckBoxPreference

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/xml/chkbox.xml -->
   <PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
         android:key="flight_columns_pref"
         android:title="Flight Search Preferences"
         android:summary="Set Columns for Search Results">
   <CheckBoxPreference
         android:key="show_airline_column_pref"
         android:title="Airline"
         android:summary="Show Airline column" />
   <CheckBoxPreference
         android:key="show_departure_column_pref"
         android:title="Departure"
         android:summary="Show Departure column" />
   <CheckBoxPreference
         android:key="show_arrival_column_pref"
         android:title="Arrival"
         android:summary="Show Arrival column" />
   <CheckBoxPreference
         android:key="show_total_travel_time_column_pref"
```

```
        android:title="Total Travel Time"
        android:summary="Show Total Travel Time column" />
  <CheckBoxPreference
        android:key="show_price_column_pref"
        android:title="Price"
        android:summary="Show Price column" />

</PreferenceScreen>

// CheckBoxPreferenceActivity.java

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class CheckBoxPreferenceActivity extends PreferenceActivity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      addPreferencesFromResource(R.xml.chkbox);
   }
}
```

В листинге 11.4 представлен XML-файл настроек chkbox.xml и простой класс явления, загружающий этот файл при помощи addPreferencesFromResource(). Как видите, в пользовательском интерфейсе есть пять флажков, каждый из которых представлен узлом CheckBoxPreference в XML-файле настроек. Каждый флажок также имеет key, который в конечном итоге применяется для сохранения состояния элемента пользовательского интерфейса, когда для этого приходит время. Когда пользователь задает состояние, CheckBoxPreference сохраняет выбранные настройки. Иными словами, когда пользователь ставит флажок напротив элемента управления настройками или снимает его, состояние сохраняется. В листинге 11.5 показана база данных для сохранения настроек из данного примера.

Листинг 11.5. База данных для хранения настроек (пример с использованием флажков)

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
   <boolean name="show_total_travel_time_column_pref" value="false" />
   <boolean name="show_price_column_pref" value="true" />
   <boolean name="show_arrival_column_pref" value="false" />
   <boolean name="show_airline_column_pref" value="true" />
   <boolean name="show_departure_column_pref" value="false" />
</map>
```

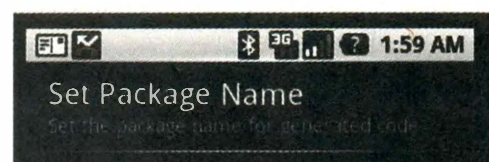
На этот раз каждая настройка снова сохраняется по ее атрибуту key. Тип данных CheckBoxPreference — boolean. Такие данные могут принимать значение true или false. True означает, что настройка выбрана, а false — что нет. Для считывания значения одной из настроек, отмеченных флажком, нужно получить доступ к совместно используемой настройке, а затем вызвать метод getBoolean(), передав ему значение key настройки:

```
Boolean option = prefs.getBoolean("show_price_column_pref", false);
```

Еще одна полезная особенность CheckBoxPreference заключается в том, что можно задавать разный текст описания (summary) в зависимости от того, выбрана настройка или нет. Соответствующие атрибуты — summaryOn и summaryOff. Теперь рассмотрим EditTextPreference.

EditTextPreference

Во фреймворке Android имеется также текстовая настройка с произвольным форматом, называемая EditTextPreference. Она может принимать необработанный текст, а не запрашивать пользовательский выбор. Ее применение можно проиллюстрировать на примере программы, которая генерирует вместо пользователя код Java. Одной из настроек такой программы может быть задаваемое по умолчанию имя пакета, с которым будут работать сгенерированные классы. Итак, здесь от нас требуется показать пользователю поле для ввода текста, в котором можно задать имя пакета для сгенерированных классов. На рис. 11.4 приведен интерфейс, а в листинге 11.6 — соответствующий XML-файл.



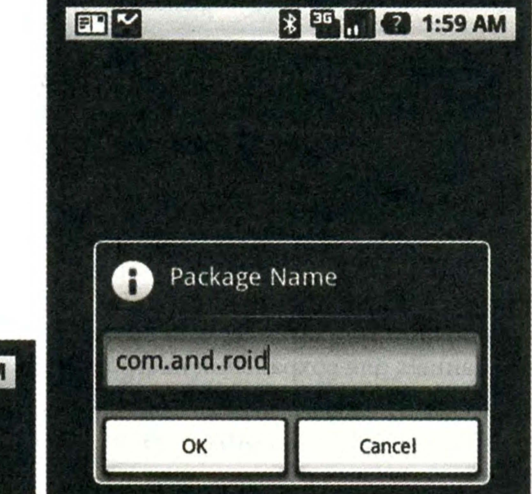


Рис. 11.4. Работа с EditTextPreference

Листинг 11.6. Пример EditTextPreference

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/xml/packagepref.xml -->
<PreferenceScreen
      xmlns:android=http://schemas.android.com/apk/res/android
         android:key="package_name_screen"
         android:title="Package Name"
         android:summary="Set package name">

   <EditTextPreference
         android:key="package_name_preference"
```

```
        android:title="Set Package Name"
        android:summary="Set the package name for generated code"
        android:dialogTitle="Package Name" />

</PreferenceScreen>

// EditTextPreferenceActivity.java

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class EditTextPreferenceActivity extends PreferenceActivity{

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      addPreferencesFromResource(R.xml.packagepref);
   }

}
```

Как видите, в листинге 11.6 определяется PreferenceScreen с одним EditTextPreference в качестве дочернего элемента. Пользовательский интерфейс, генерируемый в этом листинге, имеет PreferenceScreen слева и EditTextPreference справа (см. рис. 11.4). Когда пользователь выбирает **Set Package Name** (Задать имя пакета), система выдает диалоговое окно, в которое нужно ввести это имя. При нажатии OK настройка сохраняется в базе данных.

Как и при работе с другими настройками EditTextPreference можно получить из явления, применив параметр key. С помощью EditTextPreference можно обработать конкретный EditText, вызвав getEditText(). Например, если нужно провести валидацию, предпроцессную или послепроцессную обработку значения, которое пользователь вводит в текстовое поле. Чтобы получить текст EditTextPreference, просто используйте метод getText().

Теперь рассмотрим настройки из фреймворка RingtonePreference.

RingtonePreference

RingtonePreference предназначен для работы с рингтонами. Он применяется в программах, где пользователь в качестве одной из настроек может выбрать для телефона рингтон. На рис. 11.5 показан пользовательский интерфейс примера с RingtonePreference, а в листинге 11.7 — соответствующий XML.

Листинг 11.7. Определение настроек для RingtonePreference

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/xml/ringtone.xml -->
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
         android:key="ringtone_option_preference"
```

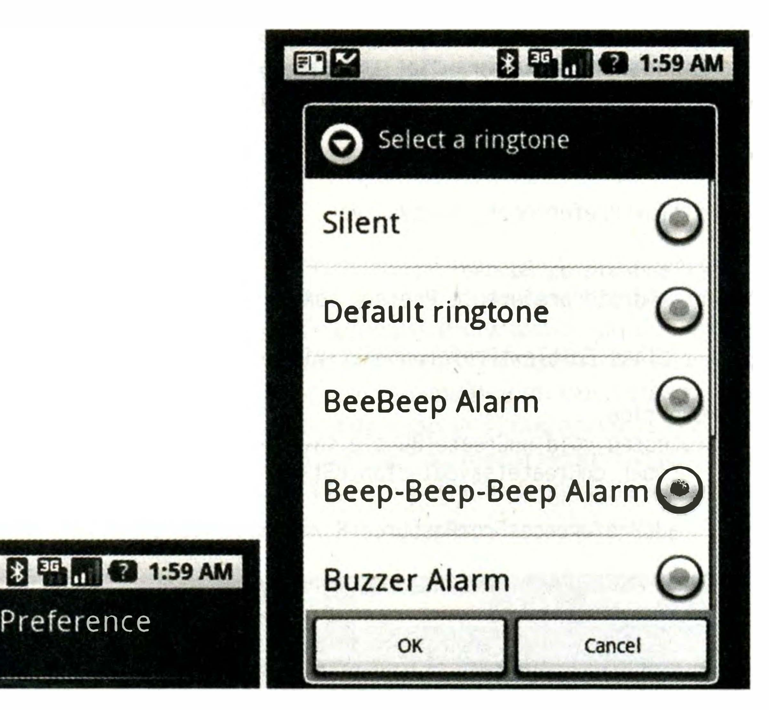


Рис. 11.5. Пример пользовательского интерфейса для RingtonePreference

```
        android:title="My Preferences"
        android:summary="Set Ring Tone Preferences">
    <RingtonePreference
        android:key="ring_tone_pref"
        android:title="Set Ringtone Preference"
        android:showSilent="true"
        android:ringtoneType="alarm"
        android:summary="Set Ringtone" />
</PreferenceScreen>

// RingtonePreferenceActivity.java

import android.os.Bundle;
import android.preference.PreferenceActivity;

public class RingtonePreferenceActivity extends PreferenceActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.ringtone);
    }
}
```

Когда пользователь выбирает **Set Ringtone Preference** (Установка рингтона), фреймворк настроек отображает ListPreference, содержащий рингтоны, которые

загружены в устройстве (см. рис. 11.5). Пользователь может выбрать рингтон, а затем нажать OK или Cancel (Отмена). При нажатии OK сделанный выбор вносится в базу данных настроек. Обратите внимание: при работе с рингтонами значением, сохраняемым в хранилище данных, является URI выбранного рингтона — за исключением случая, в котором выбирается беззвучный режим (Silent). Тогда в качестве значения сохраняется пустая строка. Пример URI выглядит так:

```
<string name="ring_tone_pref">content://media/internal/audio/media/26</string>
```

ПРИМЕЧАНИЕ

Если в эмуляторе мало рингтонов, вы можете сами добавить их. Скопируйте на карту памяти несколько музыкальных файлов (этот вопрос был рассмотрен в главе 9), перейдите в программу Android Music Player, выберите музыкальный файл, нажмите Menu (Меню) и выберите Use as ringtone (Использовать в качестве рингтона).

Наконец, необходимо отметить, что RingtonePreference, показанный в листинге 11.7, построен по тому же принципу, что и другие настройки, которые мы определяли выше. Разница заключается в том, что мы устанавливаем некоторые особые атрибуты, в том числе showSilent и ringtoneType. showSilent применяется для включения в список рингтонов беззвучного режима, а ringtoneType — для ограничения количества типов рингтонов, отображаемых в списке. Данное свойство может иметь следующие значения: ringtone, notification, alarm и all.

Организация настроек

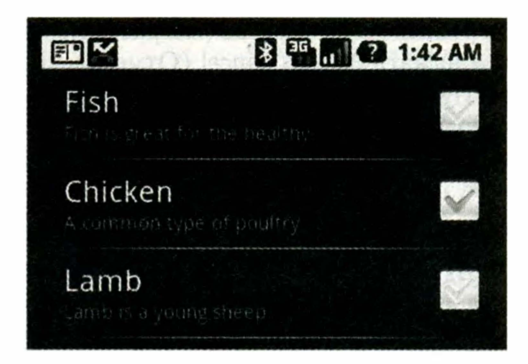
Фреймворк настроек предусматривает несколько способов распределения параметров по категориям. Если у вас очень много настроек, то можно написать вид, в котором, например, будут показаны приоритетные категории настроек. Затем пользователь сможет развернуть каждую категорию, просмотреть и обработать параметры, относящиеся к этой группе.

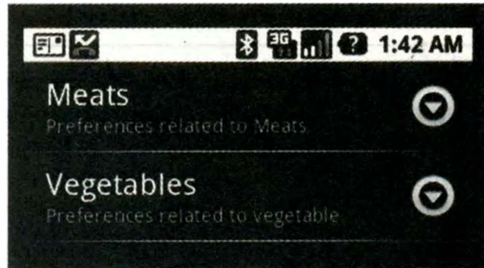
Такой механизм можно реализовать одним из двух способов. Можно ввести вложенные элементы PreferenceScreen внутри корневого PreferenceScreen либо использовать элементы PreferenceCategory — результат получается схожим. На рис. 11.6 и в листинге 11.8 показано, как применить первый способ, сгруппировав настройки при помощи вложенных элементов PreferenceScreen. В виде, приведенном слева на рис. 11.6, два экрана настроек. Один из них называется Meats (Мясные закуски), а другой — Vegetables (Овощи). При нажатии группы вы переходите к списку настроек, доступных для этой группы. В листинге 11.8 вы видите, как создаются вложенные экраны.

Листинг 11.8. Вложение элементов PreferenceScreen для организации настроек

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
     xmlns:android="http://schemas.android.com/apk/res/android"
        android:key="using_categories_in_root_screen"
        android:title="Categories"
        android:summary="Using Preference Categories">

  <PreferenceScreen
```





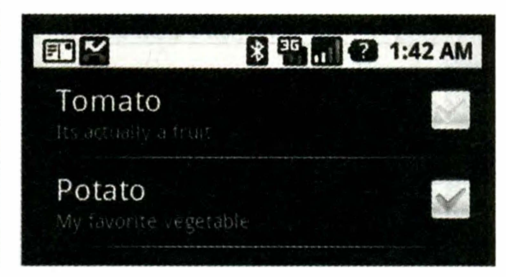


Рис. 11.6. Создание групп настроек методом вложения элементов PreferenceScreen

```
      xmlns:android="http://schemas.android.com/apk/res/android"
         android:key="meats_screen"
         android:title="Meats"
         android:summary="Preferences related to Meats">

      <CheckBoxPreference
         android:key="fish_selection_pref"
         android:title="Fish"
         android:summary="Fish is great for the healthy" />
      <CheckBoxPreference
         android:key="chicken_selection_pref"
         android:title="Chicken"
         android:summary="A common type of poultry" />
      <CheckBoxPreference
         android:key="lamb_selection_pref"
         android:title="Lamb"
         android:summary="Lamb is a young sheep" />

</PreferenceScreen>
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
         android:key="vegi_screen"
         android:title="Vegetables"
         android:summary="Preferences related to vegetable">
      <CheckBoxPreference
         android:key="tomato_selection_pref"
         android:title="Tomato "
```

```
        android:summary="It's actually a fruit" />
    <CheckBoxPreference
        android:key="potato_selection_pref"
        android:title="Potato"
        android:summary="My favorite vegetable" />

  </PreferenceScreen>

</PreferenceScreen>
```

На рис. 11.6 при создании групп применяется вложение элементов PreferenceScreen в корневом PreferenceScreen. Такой метод организации настроек удобен, если настроек у вас много и пользователю приходится прокручивать список, чтобы найти нужную настройку. Подобных ситуаций следует избегать. Если настроек не очень много, но вы тем не менее хотите объединить их в более крупные категории, можно применить PreferenceCategory — второй упомянутый нами метод. Подробности показаны на рис. 11.7 и в листинге 11.9.

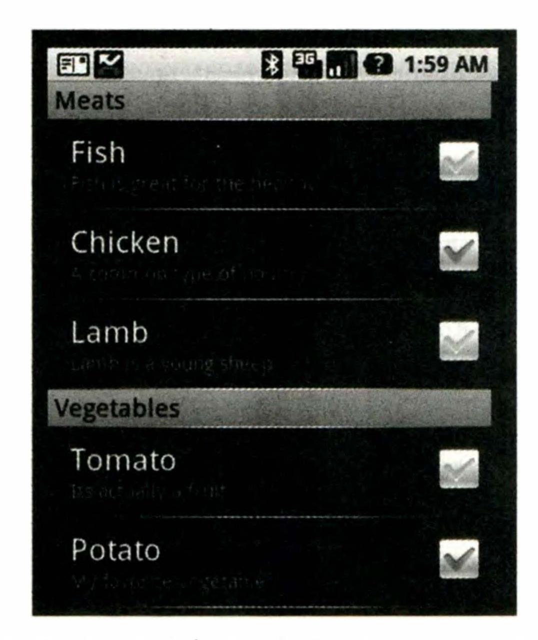


Рис. 11.7. Применение PreferenceCategory для организации настроек

На рис. 11.7 приведены те же группы, что мы использовали в нашем предыдущем примере, но теперь они организованы по категориям. Вся разница между XML в листингах 11.9 и 11.8 состоит в том, что в первом случае для вложенных категорий создаются PreferenceCategory, а не вложенные элементы PreferenceScreen.

Листинг 11.9. Создание категорий настроек

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
        android:key="using_categories_in_root_screen"
```

```
        android:title="Categories"
        android:summary="Using Preference Categories">

    <PreferenceCategory
      xmlns:android="http://schemas.android.com/apk/res/android"
        android:key="meats_category"
        android:title="Meats"
        android:summary="Preferences related to Meats">

      <CheckBoxPreference
        android:key="fish_selection_pref"
        android:title="Fish"
        android:summary="Fish is great for the healthy" />
      <CheckBoxPreference
        android:key="chicken_selection_pref"
        android:title="Chicken"
        android:summary="A common type of poultry" />
      <CheckBoxPreference
        android:key="lamb_selection_pref"
        android:title="Lamb"
        android:summary="Lamb is a young sheep" />

    </PreferenceCategory>
    <PreferenceCategory
      xmlns:android="http://schemas.android.com/apk/res/android"
        android:key="vegi_category"
        android:title="Vegetables"
        android:summary="Preferences related to vegetable">
      <CheckBoxPreference
        android:key="tomato_selection_pref"
        android:title="Tomato "
        android:summary="It's actually a fruit" />
      <CheckBoxPreference
        android:key="potato_selection_pref"
        android:title="Potato"
        android:summary="My favorite vegetable" />

    </PreferenceCategory>

</PreferenceScreen>
```

Резюме

В этой главе мы поговорили об управлении настройками в Android. Мы показали, как работать с ListPreference, CheckBoxPreference, EditTextPreference и RingtonePreference. Кроме того, мы рассмотрели, как управлять настройками при помощи программирования, а потом научились группировать настройки.

12 Изучение живых каталогов

В главе 10 мы подробно рассмотрели интерфейс OpenGL, применяемый в Android. В главе 11 было описано, как в Android происходит управление настройками программ. А в этой главе предлагаем заняться еще одной непростой темой: живыми каталогами в Android (live folders).

Живые каталоги появились в версии SDK 1.5. Они позволяют разработчику размещать поставщики содержимого (контакты, заметки, медиа) на стандартном экране устройства (который можно сравнить с рабочим столом локального компьютера или с *домашней страницей*). Когда поставщик содержимого Android, например contacts, выносится на основной экран в качестве живого каталога, то каталог будет автоматически обновляться по мере добавления, удаления контактов или их изменения в базе данных. Мы расскажем, как именно действуют такие каталоги, как внедрить их в систему и как «оживить».

Изучение живых каталогов

Живой каталог Android выполняет по отношению к поставщику содержимого ту же функцию, что и RSS-ридер по отношению к сайту, публикующему новости. В главе 3 мы уже говорили о том, что поставщики содержимого сходны с веб-сайтами тем, что предоставляют информацию по URI. По мере роста размеров сайтов и одновременного увеличения их количества, а также из-за того, что каждый сайт публиковал информацию по-своему, появилась необходимость агрегации данных со многих сайтов, чтобы читатели могли использовать для отслеживания событий одну программу-ридер. Так появились RSS. RSS помог увидеть общие принципы в построении разрозненных наборов данных. Если получен общий образец, то можно один раз написать RSS и читать с его помощью любой контент, если этот контент имеет единообразную структуру.

Принципиально живые каталоги не очень отличаются от упомянутого инструмента. RSS имеет унифицированный интерфейс, в котором публикуется контент из веб. Такой же унифицированный интерфейс живого каталога применяется для получения информации от поставщиков содержимого Android. Если поставщик содержимого удовлетворяет используемому протоколу, Android помещает ярлык живого каталога на основном экране устройства и с помощью этого ярлыка открывает доступ к данным поставщика содержимого. Когда пользователь нажимает ярлык конкретного поставщика содержимого, система связывается с этим поставщиком. Затем поставщик содержимого должен возвратить курсор. В соответствии

с контрактом для живых каталогов, курсор должен иметь заданный набор колонок. Затем происходит визуальное представление курсора — для этого применяется ListView или GridView.

Итак, если взять за основу идею общего формата, работу живого каталога можно описать так.

1. Сначала на рабочем столе создается ярлык, обозначающий набор строк, полученных от поставщика содержимого. Чтобы установить такое соединение, нужно указать вместе с ярлыком соответствующий URI.
2. Когда пользователь нажимает ярлык, система берет URI и с его помощью вызывает поставщик содержимого. Поставщик содержимого возвращает набор строк с курсором.
3. Если живой каталог ожидает получить набор колонок, который содержится в курсоре (например, название, описание и указание программы, которая должна быть активирована при нажатии строки), система представит строки в виде ListView или GridView.
4. Поскольку ListView и GridView способны обновлять свои наборы данных при изменении информации, хранящейся в базе данных, эти виды также называются живыми — так как относятся к живым каталогам.

В работе живых каталогов используется два основных принципа. Первый — названия колонок во всех курсорах являются одинаковыми. Поэтому Android может обрабатывать курсоры, направляемые к живым каталогам, одинаковым образом. Второй принцип заключается в том, что виды Android знают, как искать обновления в базовых данных курсора и соответствующим образом видоизменяться. Второй принцип применим не только к живым видам, но и практически ко всем другим видам, применяемым в пользовательском интерфейсе Android, особенно к тем, чья работа обеспечивается курсорами.

Теперь, когда мы в общих чертах описали природу живых каталогов, мы исследуем весь связанный с ними фреймворк. Описание будет разбито на два основных подраздела. В первом мы рассмотрим, как конечный пользователь работает с живым каталогом. Так мы сможем лучше понять эти необычные каталоги.

Во втором подразделе будет показано, как правильно построить живой каталог — чтобы он действительно был живым. Чтобы каталог «ожил», требуется выполнить определенную дополнительную работу, поэтому мы исследуем этот не самый очевидный аспект отдельно.

Живые каталоги с точки зрения пользователя

Живые каталоги предоставляются конечному пользователю на главном экране устройства. Пользователь работает с живым каталогом так.

1. Выходит на главный экран устройства.
2. Переходит в контекстное меню главного экрана. Чтобы открыть контекстное меню, нужно сделать длинный щелчок где-нибудь в свободной части главного экрана.
3. Находит в контекстном меню команду **Folders** (Каталоги) и щелкает на ней для просмотра доступных живых каталогов.

4. Выбирает из списка каталог и щелкает на его названии, чтобы ярлык этого каталога попал на главный экран. На главном экране создается соответствующий значок.
5. Щелкает на ярлыке живого каталога, установленного в шаге 4, чтобы вывести строки с информацией (данные, находящиеся в этом живом каталоге) в ListView или GridView.
6. Щелкает на одной из строк для активации программы, которой известно, как отобразить такую строку с данными.
7. Использует другие параметры меню, отображаемые программой, чтобы просмотреть желаемый элемент или управлять им. Кроме того, этот элемент меню программы можно использовать для создания любых новых элементов, допустимых в этой программе.

Необходимо также отметить, что при отображении живых каталогов автоматически указываются любые изменения, применяемые к элементу или набору элементов.

Рассмотрим все эти этапы, проиллюстрировав их при помощи скриншотов. Начнем с этапа 1. Перед нами типичный главный экран Android (рис. 12.1). Обратите внимание — внешний вид этого экрана может немного отличаться в зависимости от того, какой версией Android вы пользуетесь.

Если сделать на этой странице длинный щелчок, то появится контекстное меню (рис. 12.2).



Рис. 12.1. Главный экран Android



Рис. 12.2. Контекстное меню главной страницы Android

Если щелкнуть на команде Folders (Каталоги), то Android откроет еще одно меню, в котором будут представлены доступные живые каталоги (рис. 12.3). Созданием живого каталога мы займемся в следующем разделе, а пока предположим, что нужный нам живой каталог уже создан и назван New live folder (Новый живой каталог).

При щелчке на строке New Live Folder (Новый живой каталог) Android создаст на главном экране ярлык, представляющий живой каталог. В нашем примере это будет Contacts LF (ЖК Контакты). LF (ЖК) в данном случае — это сокращение от Live Folder (Живой каталог) (рис. 12.4). В этом каталоге будут отображаться контакты, хранимые в соответствующей базе данных. (О том, как называть каталог, мы поговорим позже, когда будем описывать класс `AllContactsLiveFolderCreatorActivity`.)

В следующем разделе будет показано, что за создание Contacts LF (ЖК Контакты) отвечает специальное явление. Поскольку в данном случае речь идет о работе пользователя с каталогом, можно поставить себя на место пользователя: нажать ярлык Contacts LF (ЖК Контакты) и просмотреть список контактов, отображаемый в `ListView` (рис. 12.5).

В зависимости от того, сколько контактов сохранено в устройстве, список может выглядеть по-разному. Можно щелкнуть на любом контакте, чтобы просмотреть детали о нем (рис. 12.6).

Можно нажать кнопку Menu (Меню), расположенную в нижней части экрана, и просмотреть, какие операции можно совершать с отдельно взятым контактом (рис. 12.7).

Пр щелчке на кнопке изменения контакта откроется экран, изображенный на рис. 12.8.

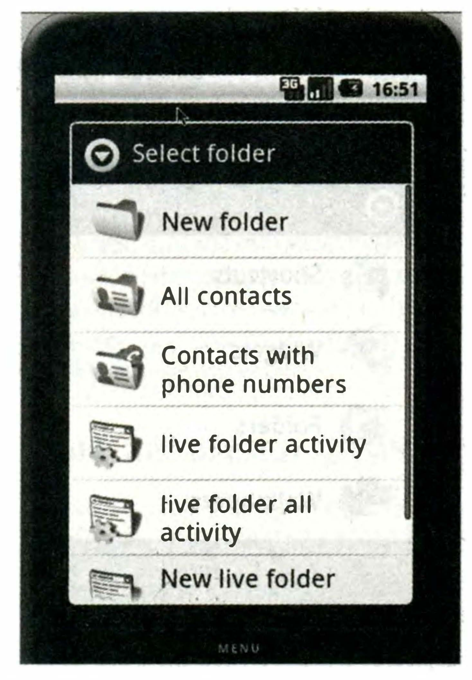


Рис. 12.3. Просмотр списка доступных живых каталогов

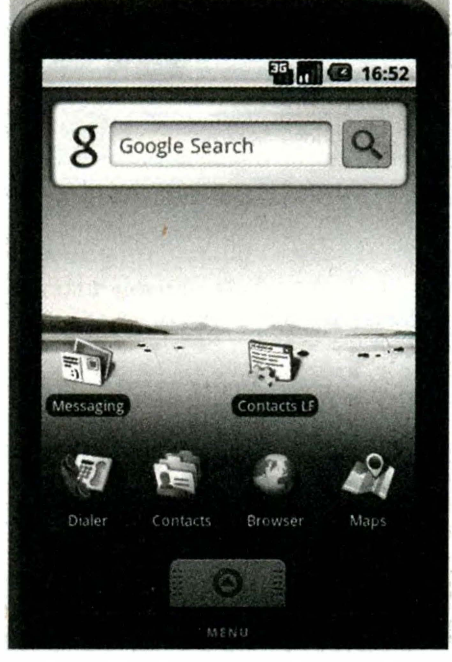


Рис. 12.4. Ярлык живого каталога на главном экране

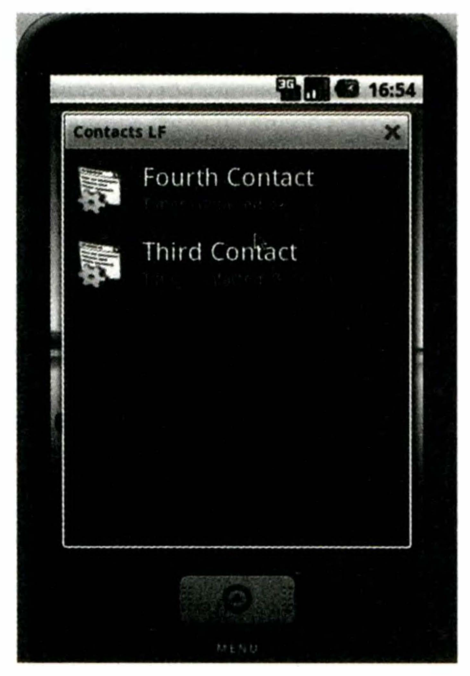


Рис. 12.5. Показ живого каталога с контактами

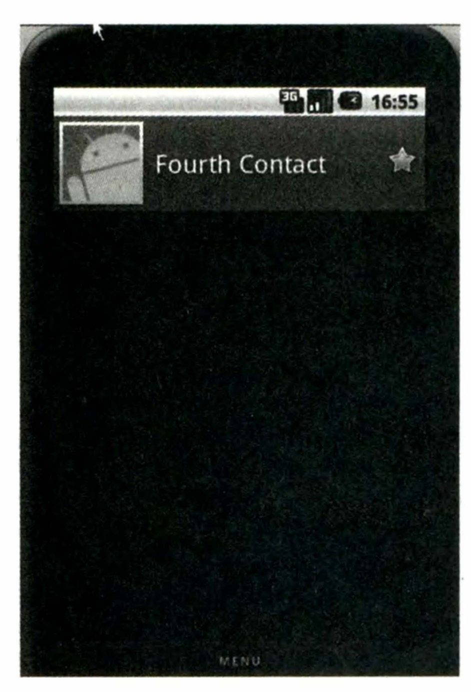


Рис. 12.6. Открытие живого каталога с контактами

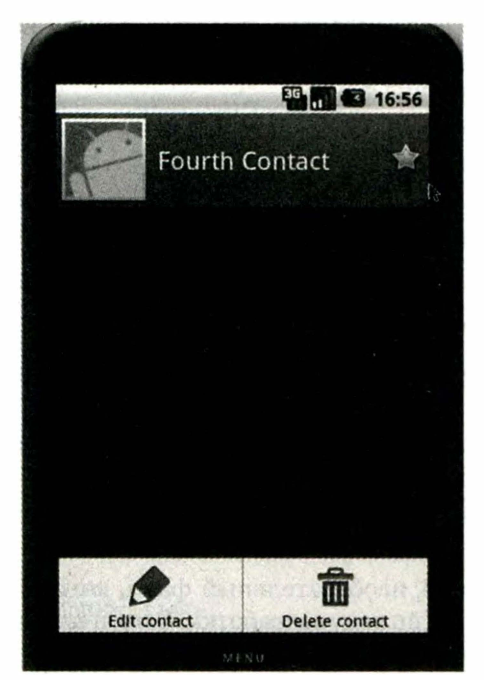


Рис. 12.7. Меню с параметрами для отдельно взятого контакта



Рис. 12.8. Изменение контактной информации

Чтобы познакомиться с «живой» составляющей данного каталога, можно удалить этот контакт или создать новый. После возвращения к виду, где показан Contacts LF (ЖК Контакты), вы увидите, что внесенные изменения вступили в силу. Чтобы проследить ход изменений, можно несколько раз подряд нажать кнопку Back (Назад).

Создание живого каталога

Теперь, рассказав, что такое живые каталоги и для чего они нужны, мы покажем, как создать такой каталог. После создания живого каталога для него можно делать ярлык на главном экране устройства. Кроме того, рассмотрим те функции каталога, которые и делают его живым.

Для создания живого каталога нужны две вещи: явление и специальный поставщик содержимого. Android использует название (этикетку) этого явления для заполнения списка имеющихся каталогов, как это показано на рис. 12.3. Кроме того, Android активирует явление, чтобы получить URI, по которому уже будет получен список строк для отображения.

URI, сообщаемый в явлении, должен указывать на выделенный поставщик содержимого, который предназначен для возвращения строк. Поставщик содержимого возвращает эти строки посредством «правильного» курсора. Мы называем курсор «правильным», так как он должен иметь заранее известную заданную схему названий колонок.

Обычно два этих элемента упаковываются в программе, которая затем развертывается в устройстве. Кроме того, чтобы все это работало, нам понадобится несколько вспомогательных файлов. Мы объясним и продемонстрируем эти идеи на примере, в котором будут содержаться следующие файлы:

- AndroidManifest.xml — в этом файле определяется, какое явление нужно вызвать, чтобы создать определение живого каталога;
- AllContactsLiveFolderCreatorActivity.java — это явление предоставляет определение для живого каталога, в котором можно будет отображать все контакты из базы данных устройства;
- MyContactsProvider.java — данный поставщик содержимого будет отвечать на URI живого каталога, возвращающий курсор с контактами. Внутри системы этот поставщик содержимого использует поставщик контактов, входящий в состав Android;
- MyCursor.java — это специальный курсор, который умеет делать повторные запросы (requery) при изменении базовых данных;
- BetterCursorWrapper.java — этот файл требуется MyCursor для организации повторных запросов;
- SimpleActivity.java — это простое явление, необязательный файл, который можно применять для тестирования проекта по ходу разработки. В окончательной версии программы этот файл не требуется.

Чтобы вы поняли, как работают живые каталоги, мы опишем каждый из этих файлов.

AndroidManifest.xml

AndroidManifest.xml вам уже знаком: такой файл присутствует в любой программе Android. Раздел файла, описывающий живые каталоги, отмечен как комментарий, и в нем мы помещаем явление AllContactsLiveFolderCreatorActivity, при помощи которого создается живой каталог (листинг 12.1). Этот факт выражается при помощи объявления намерения с действием android.intent.action.CREATE_LIVE_FOLDER.

Название явления **New live folder** (Новый живой каталог) будет отображаться в контекстном меню главной страницы (см. рис. 12.3). Как было объяснено в подразделе «Живые каталоги с точки зрения пользователя», чтобы попасть в контекстное меню с главной страницы, нужно сделать длинный щелчок.

Листинг 12.1. Файл AndroidManifest.xml для определения живого каталога

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.ai.android.livefolders"
      android:versionCode="1"
      android:versionName="1.0">
   <application android:icon="@drawable/icon"
                android:label="@string/app_name">
        <activity android:name=".SimpleActivity"
                android:label="@string/app_name">
           <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name=
                 "android.intent.category.LAUNCHER" />
           </intent-filter>
        </activity>

      <!-- ЖИВЫЕ КАТАЛОГИ -->
        <activity
           android:name=".AllContactsLiveFolderCreatorActivity"
           android:label="New live folder "
           android:icon="@drawable/icon">

           <intent-filter>
              <action android:name=
                    "android.intent.action.CREATE_LIVE_FOLDER" />
              <category android:name=
                    "android.intent.category.DEFAULT" />
           </intent-filter>
        </activity>

      <provider android:authorities="com.ai.livefolders.contacts"
      android:multiprocess="true"
           android:name=".MyContactsProvider" />

   </application>
   <uses-sdk android:minSdkVersion="3" />
<uses-permission android:name="android.permission.READ_CONTACTS">
</uses-permission>
</manifest>
```

В коде из листинга 12.1 среди прочего необходимо заострить внимание на объявлении поставщика (provider), связанном с URI content://com.ai.livefolders.contacts и обслуживаемом классом поставщика MyContactsProvider. Этот класс предоставляет курсор для заполнения ListView, открывающегося при нажатии на ярлык соответствующего живого каталога (см. рис. 12.5). Явление живого каталога AllContactsLiveFolderCreatorActivity должно знать этот URI и возвращать его Android при активации. Android активирует это явление при выборе названия живого каталога и с его помощью создает ярлык для каталога на главном экране.

В соответствии с протоколом живого каталога намерение CREATE_LIVE_FOLDER позволяет контекстному меню с главной страницы отображать AllContactsLiveFolderCreatorActivity как команду с названием **New live folder** (Новый живой каталог) (см. рис. 12.3). При выборе этой команды меню на домашней странице создается ярлык; эта операция показана на рис. 12.4.

AllContactsLiveFolderCreatorActivity должно определять этот ярлык, состоящий из изображения и этикетки. В нашем случае этикетка в коде AllContactsLiveFolder CreatorActivity — это Contacts LF. Итак, рассмотрим исходный код, используемый при создании живого каталога.

AllContactsLiveFolderCreatorActivity.java

Класс AllContactsLiveFolderCreatorActivity выполняет одну функцию: генерирует (то есть создает) живой каталог (листинг 12.2). Можете считать его шаблоном живого каталога. Всякий раз при щелчке на этом явлении (команда **Folders** (Каталоги) в контекстном меню главной страницы) будет создаваться живой каталог на главной странице.

Для выполнения своей функции явление сообщает активатору — в данном случае домашней странице или фреймворку живых каталогов — название живого каталога, изображение, которое будет использоваться в ярлыке, URI, по которому доступны данные, и способ отображения (список или сетка). Фреймворк, в свою очередь, отвечает за создание ярлыка живого каталога на домашней странице.

ПРИМЕЧАНИЕ

Все контракты, необходимые для работы живого каталога, описаны в документации по Android SDK, в разделе о классе android.provider.LiveFolders.

Листинг 12.2. Исходный код AllContactsLiveFolderCreatorActivity

```
public class AllContactsLiveFolderCreatorActivity extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);

      final Intent intent = getIntent();
      final String action = intent.getAction();

      if (LiveFolders.ACTION_CREATE_LIVE_FOLDER.equals(action)) {
           setResult(RESULT_OK,
              createLiveFolder(MyContactsProvider.CONTACTS_URI,
                 "Contacts LF",
```

```
                R.drawable.icon)
            );
      }

      else {
         setResult(RESULT_CANCELED);
      }
      finish();
   }

   private Intent createLiveFolder(Uri uri, String name, int icon)
   {
      final Intent intent = new Intent();
      intent.setData(uri);
      intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_NAME, name);
      intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_ICON,
            Intent.ShortcutIconResource.fromContext(this, icon));
      intent.putExtra(LiveFolders.EXTRA_LIVE_FOLDER_DISPLAY_MODE,
            LiveFolders.DISPLAY_MODE_LIST);
      return intent;
   }
}
```

Метод createLiveFolder передает значения намерению, которое его активирует. После того как намерение вернется к вызывающей стороне, оно сообщит следующие данные:

- название живого каталога;
- изображение, которое будет использоваться в ярлыке;
- способ отображения: список или сетка;
- данные или контент URI для активации данных.

Эта информация важна для создания ярлыка живого каталога (см. рис. 12.4). Когда пользователь нажимает этот ярлык, система вызывает URI для поиска данных. Поставщик содержимого, определяемый в этом URI, предоставляет стандартный курсор. Мы покажем вам код для этого поставщика содержимого: класс MyContactsProvider.

MyContactsProvider.java

MyContactsProvider выполняет следующие задачи.

1. Идентифицирует входящий URI, который выглядит так: content://com.ai.livefolders.contacts/contacts.
2. Делает внутрисистемный вызов поставщика содержимого для работы с контактами, который входит в состав Android и имеет URI content://contacts/people/.
3. Построчно считывает информацию из курсора и сопоставляет ее с информацией другого курсора, например MatrixCursor, чтобы названия колонок совпадали с требованиями фреймворка живых каталогов.
4. «Заворачивает» MatrixCursor в другой курсор, чтобы повторные запросы (requery) к завернутому таким образом курсору при необходимости сопровождались вызовами поставщика контактов.

Код MyContactsProvider приведен в листинге 12.3. Наиболее важные его части выделены полужирным.

Листинг 12.3. Исходный код MyContactsProvider

```
public class MyContactsProvider extends ContentProvider {

   public static final String AUTHORITY = "com.ai.livefolders.contacts";

   // Uri, служащий для ввода при создании живого каталога
   public static final Uri CONTACTS_URI = Uri.parse("content://" +
         AUTHORITY + "/contacts" );

   // код, чтобы отличать этот URI
   private static final int TYPE_MY_URI = 0;
   private static final UriMatcher URI_MATCHER;
   static{
      URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
      URI_MATCHER.addURI(AUTHORITY, "contacts", TYPE_MY_URI);
   }

   @Override
   public boolean onCreate() {
      return true;
   }

   @Override
   public int bulkInsert(Uri arg0, ContentValues[] values) {
      return 0; // ничего не вставляем
   }

   // Набор колонок, необходимых для работы живого каталога.
   // Контракт для живого каталога.
   private static final String[] CURSOR_COLUMNS = new String[]
   {
      BaseColumns._ID,
      LiveFolders.NAME,
      LiveFolders.DESCRIPTION,
      LiveFolders.INTENT,
      LiveFolders.ICON_PACKAGE,
      LiveFolders.ICON_RESOURCE
   };

   // В случае отсутствия строк
   // эта вставка используется для вывода сообщения об ошибке.
   // Обратите внимание - вставка имеет такой же набор колонок,
   // как и живой каталог.
   private static final String[] CURSOR_ERROR_COLUMNS = new String[]
   {
      BaseColumns._ID,
```

```
      LiveFolders.NAME,
      LiveFolders.DESCRIPTION
   };

   // строка с сообщением об ошибке
   private static final Object[] ERROR_MESSAGE_ROW =
      new Object[]
      {
         -1,                              // id
         "No contacts found",             //имя
         "Check your contacts database"  //описание
      };

   // курсор для работы с ошибками
   private static MatrixCursor sErrorCursor = new ➥
                  MatrixCursor(CURSOR_ERROR_COLUMNS);
   static
   {
      sErrorCursor.addRow(ERROR_MESSAGE_ROW);
   }

   // Колонки, которые необходимо получить из базы данных с контактами.
   private static final String[] CONTACTS_COLUMN_NAMES = new String[]
   {
      People._ID,
      People.DISPLAY_NAME,
      People.TIMES_CONTACTED,
      People.STARRED
   };

   public Cursor query(Uri uri, String[] projection, String selection,
                       String[] selectionArgs, String sortOrder)
   {
      // определение uri и возвращение ошибки в случае несовпадения
      int type = URI_MATCHER.match(uri);
      if(type == UriMatcher.NO_MATCH)
      {
         return sErrorCursor;
      }

      Log.i("ss", "query called");

      try
      {
         MatrixCursor mc = loadNewData(this);
         mc.setNotificationUri(getContext().getContentResolver(),
                     Uri.parse("content://contacts/people/"));
         MyCursor wmc = new MyCursor(mc,this);
         return wmc;
      }
```

```
    catch (Throwable e)
    {
        return sErrorCursor;
    }
}

public static MatrixCursor loadNewData(ContentProvider cp)
{
    MatrixCursor mc = new MatrixCursor(CURSOR_COLUMNS);
     Cursor allContacts = null;
     try
     {
        allContacts = cp.getContext().getContentResolver().query(
            People.CONTENT_URI,
            CONTACTS_COLUMN_NAMES,
            null, // фильтр строк
            null,
            People.DISPLAY_NAME); // упорядочить по

        while(allContacts.moveToNext())
        {
            String timesContacted =
                "Times contacted: "+allContacts.getInt(2);

            Object[] rowObject = new Object[]
            {
                allContacts.getLong(0),          //id
                allContacts.getString(1),        //имя
                timesContacted,                  //описание
                Uri.parse("content://contacts/people/"
                    +allContacts.getLong(0)),    //uri намерения
                cp.getContext().getPackageName(), //пакет
                R.drawable.icon                  //ярлык
            };
            mc.addRow(rowObject);
        }
        return mc;
    }
    finally
    {
        allContacts.close();
    }
}

@Override
public String getType(Uri uri)
{
    // Указание типа MIME для заданного URI,
    // предназначенного для обертывающего поставщика,
    // обычно имеет вид
    // "vnd.android.cursor.dir/vnd.google.note".
    return People.CONTENT_TYPE;
```

```
    }

    public Uri insert(Uri uri, ContentValues initialValues) {
        throw new UnsupportedOperationException(
            "no insert as this is just a wrapper");
    }

    @Override
    public int delete(Uri uri, String selection,
                          String[] selectionArgs) {
        throw new UnsupportedOperationException(
        "no delete as this is just a wrapper");
    }

    public int update(Uri uri, ContentValues values,
            String selection, String[] selectionArgs)
    {
        throw new UnsupportedOperationException(
        "no update as this is just a wrapper");
    }
}
```

В листинге 12.4 приведен стандартный набор колонок, необходимый для работы живого каталога.

Листинг 12.4. Колонки, необходимые для выполнения контракта живого каталога

```
private static final String[] CURSOR_COLUMNS = new String[]
{
    BaseColumns._ID,
    LiveFolders.NAME,
    LiveFolders.DESCRIPTION,
    LiveFolders.INTENT,
    LiveFolders.ICON_PACKAGE,
    LiveFolders.ICON_RESOURCE
};
```

Назначение большинства из них является очевидным, кроме элемента INTENT. На рис. 12.5 вы видите, что NAME — это заголовок элемента из списка. DESCRIPTION будет находиться ниже NAME в том же элементе списка.

Поле INTENT содержит строковые значения и указывает URI, который элемент имеет в поставщике содержимого. Для действия VIEW, выполняемого при щелчке пользователя на элементе, Android использует этот URI. Вот почему поле со строковыми значениями называется INTENT — ведь внутри системы Android получает намерение (англ. intent) из строкового URI.

Последние две строки относятся к ICON, который отображается как часть списка. Вновь отсылаем вас к рис. 12.5, где изображены ярлыки (icons). Внимательно изучите листинг 12.3, чтобы понять, как в колонках предоставляются значения из базы данных с контактами.

Необходимо также отметить, что указанный выше MyContactsContentProvider (поставщик-обертка) выполняет код из листинга 12.5, чтобы сообщить базовому курсору о необходимости отслеживания всех изменений данных.

Листинг 12.5. Регистрация URI с курсором

```
MatrixCursor mc = loadNewData(this);
mc.setNotificationUri(getContext().getContentResolver(),
                Uri.parse("content://contacts/people/"));
```

Функция `loadNewData()` получает список контактов от поставщика содержимого и создает `MatrixCursor`, который имеет колонки, показанные в листинге 12.4. Затем код приказывает `MatrixCursor` зарегистрироваться с `ContentResolver`, чтобы `ContentResolver` мог предупреждать курсор о любых изменениях данных, на которые указывает URI (`content://contacts/people`).

Вам будет интересно узнать, что URI для отслеживания совпадает не с URI `MyContactsProvider`, а с URI поставщика контактов, интегрированного в Android. Это объясняется тем, что `MyContactsProvider` служит просто оболочкой «настоящего» поставщика содержимого. То есть курсор должен следить за основным поставщиком содержимого, а не за оболочкой.

Кроме того, важно «завернуть» `MatrixCursor` в наш собственный курсор, как это показано в листинге 12.6

Листинг 12.6. Создание оболочки для курсора

```
MatrixCursor mc = loadNewData(this);
mc.setNotificationUri(getContext().getContentResolver(),
                Uri.parse("content://contacts/people/"));
MyCursor wmc = new MyCursor(mc,this);
```

Чтобы понять, каким образом обертывается курсор, нужно разобраться, как в видах обновляется измененный контент. Поставщик содержимого, например Contacts (Контакты), обычно сообщает курсору о необходимости отслеживать изменения. И для этого при реализации метода `query` регистрируется URI. Это делается при помощи `cursor.setNotificationUri`. Затем курсор регистрирует этот URI и все его дочерние URI с поставщиком содержимого. После этого, когда в данных поставщика содержимого происходит вставка или удаление информации, код вставки или удаления должен инициировать событие, обозначающее внесение изменений в строки, которым соответствует конкретный URI.

Таким образом, инициируется обновление курсора через `requery` и соответствующее обновление вида. К сожалению, `MatrixCursor` не приспособлен для работы с такими повторными запросами. `SQLiteCursor`, напротив, предназначен для этого, но в данном случае мы не можем применить `SQLiteCursor`, так как соотносим имеющиеся колонки с новым набором колонок.

Для обхода этого ограничения мы обертываем `MatrixCursor` в оболочку курсора и переопределяем метод `requery`, чтобы сбросить внутренний `MatrixCursor` и создать новый, с актуальными данными. Если выразиться точнее, то при каждом изменении данных нам нужен новый `MatrixCursor`. Но с точки зрения фреймворка Android для работы с живыми каталогами возвращается только «обернутый» внешний курсор. Таким образом, фреймворку живых каталогов сообщается, что курсор только один, но под его прикрытием при изменении данных передаются новые курсоры.

Этот механизм иллюстрируют два следующих класса.

MyCursor.java

Рассмотрим, как сначала `MyCursor` инициализируется при помощи `MatrixCursor` (листинг 12.7). В ответ на повторный запрос `MyCursor` делает обратный вызов к по-

ставщику, чтобы вернуть MatrixCursor. Затем новый MatrixCursor заменяет старый курсор методом set.

ПРИМЕЧАНИЕ

Можно предположить, что для достижения того же эффекта нужно было переопределить повторный запрос (requery) MatrixCursor. Но в этом классе отсутствует метод, который позволил бы удалить старые данные и начать все сначала. По этой причине описанный выше обходной маневр вполне оправдан (обратите внимание — MyCursor дополняет BetterCursorWrapper, об этом мы поговорим ниже).

Листинг 12.7. Исходный код MyCursor

```
public class MyCursor extends BetterCursorWrapper
{
   private ContentProvider mcp = null;

   public MyCursor(MatrixCursor mc, ContentProvider inCp)
   {
      super(mc);
      mcp = inCp;
   }
   public boolean requery()
   {
      MatrixCursor mc = MyContactsProvider.loadNewData(mcp);
      this.setInternalCursor(mc);
      return super.requery();
   }
}
```

Теперь мы научимся обертывать курсор. Для этого рассмотрим класс BetterCursorWrapper.

BetterCursorWrapper.java

Класс BetterCursorWrapper (листинг 12.8) очень похож на класс CursorWrapper, входящий в состав фреймворка базы данных в Android. Но нам нужны еще две вещи, которых не хватает в CursorWrapper. Во-первых, в нем нет метода set для замены внутреннего курсора из метода requery. Во-вторых, CursorWrapper — это не CrossProcessCursor. Живым каталогам требуется именно CrossProcessCursor, а не обычный курсор, поскольку при работе живых каталогов не учитываются границы процессов.

Листинг 12.8. Исходный код BetterCursorWrapper

```
public class BetterCursorWrapper implements CrossProcessCursor
{
   // содержит внутренний курсор для делегирования методов
   protected CrossProcessCursor internalCursor;

   // конструктор принимает межпроцессорный курсор в качестве ввода
   public BetterCursorWrapper(CrossProcessCursor inCursor)
   {
      this.setInternalCursor(inCursor);
   }

   // можно сбросить один из методов производного класса
   public void setInternalCursor(CrossProcessCursor inCursor)
```

```
    {
        internalCursor = inCursor;
    }

    // далее следуют все делегированные методы
    public void fillWindow(int arg0, CursorWindow arg1) {
        internalCursor.fillWindow(arg0, arg1);
    }
    // ..... прочие делегированные методы
}
```

Мы не приводим здесь весь класс целиком, но при помощи Eclipse вы легко можете сгенерировать его остаток. После того как загрузите этот частичный (partial) класс в Eclipse, поставьте указатель мыши на переменную `internalCursor`. Щелкните правой кнопкой мыши и выберите **Source ▸ Generate Delegated Methods** (Источник ▸ Сгенерировать делегированные методы). Затем Eclipse достроит остаток класса за вас. Теперь рассмотрим простое явление, которое требуется нам для завершения проекта-примера.

SimpleActivity.java

`SimpleActivity.java` (листинг 12.9) — не самый важный класс для работы с живыми каталогами, но включая его в проект, мы получаем общий образец для построения всех наших проектов. Кроме того, он позволяет развертывать программу и просматривать ее на экране при отладке проекта в Eclipse.

Листинг 12.9. Исходный код SimpleActivity

```
public class SimpleActivity extends Activity
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

В данном случае можно использовать в качестве `main.xml` любой простой XML-шаблон, идентифицируемый при помощи `R.layout.main`. В листинге 12.10 показан пример.

Листинг 12.10. Простой XML-файл шаблона

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Live Folder Example"
    />
</LinearLayout>
```

Теперь у нас есть все классы, необходимые для создания, развертывания и запуска программы с живыми каталогами в Eclipse. В завершение главы о живых каталогах мы покажем, что происходит при доступе к такому каталогу.

Работа с живыми каталогами

Подготовив все эти файлы для проекта, использующего живые каталоги, мы можем создать проект и развернуть его в эмуляторе. Если вы используете Eclipse, то в эмуляторе будет отображаться простое явление. Теперь мы можем работать с живым каталогом, который создали сами.

Перейдите на главный экран устройства; он должен выглядеть как на рис. 12.1. Следуйте образцу, описанному в начале этой главы в разделе «Живые каталоги с точки зрения пользователя». В частности, нужно найти созданный живой каталог и сделать для него ярлык (см. рис. 12.4). Войдите в Contacts LF (ЖК Контакты) — и увидите, что список заполнен контактами (см. рис. 12.5).

Резюме

Живой каталог — это инновационный элемент, позволяющий при помощи единственного щелчка отображать на главной странице устройства все изменения, происходящие с данными. Эти данные могут быть практически любыми — главное, чтобы их можно было расположить в виде набора колонок, представляемых в форме списка. Все данные должны сопровождаться информацией, позволяющей идентифицировать их по имени и характеризовать при помощи описания. Практически любые информационные элементы соответствуют этим требованиям, поскольку большинство данных можно определенным образом именовать и описать. Кроме того, полезно иметь явление, в котором будут отображаться более подробные данные о том живом каталоге, на котором щелкнет пользователь. Эти данные могут быть не только локальными (например, как список контактов), но и сетевыми, как резюме прочитанных блогов.

Кроме того, в этой главе было показано, как Android при помощи специальных явлений получает контент URI для живого каталога. Затем система применяет этот URI для получения коллекции строк, которые будут отображаться как содержание конкретного живого каталога. Мы показали, как внедрить в систему поставщик содержимого, который будет давать информацию для живого каталога на базе предыдущего URI.

В этой главе также были объяснены нюансы, связанные с использованием курсоров при работе с живыми каталогами, и механизмы, при помощи которых можно выставлять в качестве источников информации для живых каталогов уже имеющиеся поставщики содержимого. Мы рассказали о курсорах-оболочках и показали, как регистрировать ContentResolver для получения обновлений информации.

В следующей главе будет описан еще один инновационный элемент домашней страницы — виджеты.

13 Виджеты основного экрана

В этой главе будут подробно рассмотрены виджеты главного экрана Android. Виджеты главного экрана, как и живые каталоги, — это еще одна возможность, позволяющая представлять часто изменяющуюся информацию на главном экране Android. С профессиональной точки зрения виджеты основного экрана являются изолированными видами (тем не менее они заполнены данными), отображаемыми на домашней странице. Содержимое этих видов обновляется с регулярными интервалами при помощи процессов, работающих в фоновом режиме.

Например, почтовый виджет главного экрана может уведомлять вас о том, сколько непрочитанных писем у вас накопилось. Однако необходимо отметить, что виджет просто покажет количество электронных писем, а не сами письма. Если вы щелкнете на количестве электронных писем, то откроется явление, в котором будут отображены сами письма. Они могут приходить даже с внешних источников — например с Yahoo!, Gmail, Hotmail, так как устройство может получать доступ к данным о количестве писем по протоколу HTTP или при помощи других механизмов сетевого взаимодействия.

Эта глава разбита на три основных раздела. В первом разделе мы сообщим вводную информацию о виджетах главного экрана и опишем их архитектуру. Мы расскажем, как Android применяет `RemoteViews` для показа виджетов и ассимилирует широковещательные приемники для обновления этих `RemoteViews`. Мы научимся создавать явления для конфигурирования виджетов главного экрана и исследуем взаимосвязи между службами и виджетами. К концу раздела мы составим четкое представление о жизненном цикле виджета главного экрана.

Во втором разделе при помощи аннотированного кода будет показано, как проектировать и разрабатывать домашний виджет. Вы узнаете, как в Android происходит определение виджетов и как программировать широковещательные приемники для обновления виджетов. Будет показано, как управлять состоянием виджета посредством совместно используемых настроек и как писать явление для конфигурации виджетов.

В третьем разделе поговорим о соответствии требованиям (suitability), ограничениях и более общих рекомендациях, связанных с использованием виджетов. В этом же разделе будут рассмотрены области и возможности применения виджетов. Здесь же будут высказаны соображения относительно программирования таких виджетов, которые нуждаются в значительно более частых обновлениях.

В завершение главы будут перечислены некоторые ресурсы, посвященные программированию виджетов.

Архитектура виджетов основного экрана

Начнем обсуждение архитектуры виджетов главного экрана с выяснения, что же такое виджет. Обсудим этот вопрос подробно.

Что такое виджеты главного экрана

Как было сказано во введении к главе, виджеты главного экрана — это виды, которые можно отображать на домашней странице и часто обновлять.

Как и в случае с видом, внешнее оформление виджета определяется в XML-файле шаблона. В случае с виджетом, кроме шаблона вида, необходимо определить, сколько места будет занимать виджет на главном экране.

При определении виджета также применяется пара классов Java, обеспечивающих инициализацию вида и его обновление с нужной частотой. Эти классы также управляют жизненным циклом виджета главного экрана. Классы отвечают на перетаскивание виджета на главный экран и на удаление виджета с экрана в корзину.

ПРИМЕЧАНИЕ

Вид и соответствующие классы Java строятся так, что в итоге оказываются изолированными друг от друга. Например, любая служба или любое явление Android способны находить вид по ID его шаблона и заполнять вид данными (именно так заполняются шаблоны), а потом посылать на главный экран. Когда вид отправляется на главный экран, он разъединяется с базовым кодом Java.

В определении виджета должна содержаться как минимум следующая информация:

- шаблон того вида, который будет отображаться на главном экране, а также указание размера вида — он должен занимать на главном экране столько места, сколько нужно. Помните, что это просто вид, в котором нет никаких данных. Обновлением вида будет заниматься класс Java;
- таймер, задающий частоту обновлений;
- класс Java, называемый «поставщиком виджета», способный реагировать на обновление данных таймера и изменять вид тем или иным образом, заполняя его данными.

Когда будет определен вид и предоставлены классы Java, виджет можно будет использовать. Сначала рассмотрим, как строится работа пользователя с виджетами.

Как пользователь воспринимает виджеты основного экрана

Набор функций виджета в Android позволяет выбирать виджет для размещения на домашней странице. Когда виджет размещен, его при необходимости можно конфигурировать с помощью специального явления. Начнем с нахождения нужного нам виджета и размещения его экземпляра на основном экране.

Создание экземпляра виджета на основном экране

Для доступа к списку доступных виджетов нужно сделать на домашней странице длинный щелчок. Таким образом откроется экранное контекстное меню, показанное на рис. 13.1.



Рис. 13.1. Контекстное меню основного экрана

При выборе виджетов из этого списка вы увидите новый экран со списком доступных виджетов (рис. 13.2).

Большинство этих виджетов входят в состав Android. В зависимости от того, с какой версией Android вы работаете, набор виджетов может различаться. В данном списке есть виджет под названием Birthday Widget, созданный нами специально для этого упражнения. Если выбрать этот виджет в списке, экземпляр такого виджета будет создан на основном экране и станет выглядеть как на рис. 13.3.

Это и есть Birthday Widget. В его заголовке будут указаны имя именинника, количество дней, оставшихся до дня рождения, дата дня рождения и ссылка для покупки подарков.

ПРИМЕЧАНИЕ

Вид, создаваемый на главной странице и определяющий этот виджет, называется экземпляром виджета (widget instance). Предполагается, что из такого определения виджета можно создать несколько подобных экземпляров.



Рис. 13.2. Список виджетов для размещения на основном экране

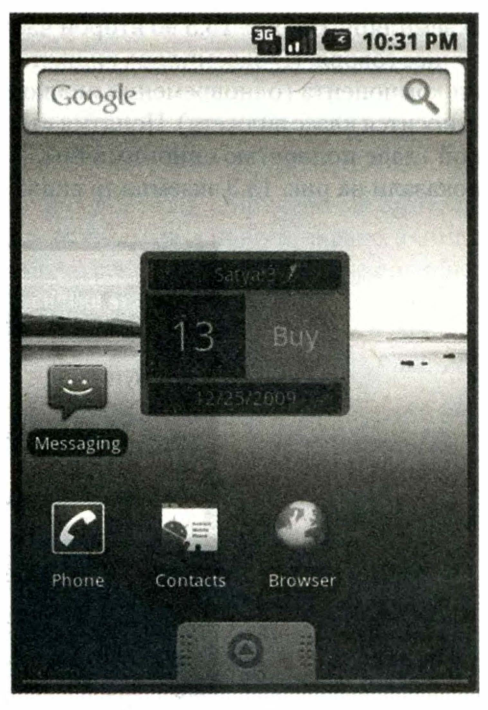


Рис. 13.3. Пример Birthday Widget

Конфигуратор виджетов

Здесь мы поговорим о том *конфигураторе виджетов*, который упоминали выше. Определение виджета может включать спецификацию явления, которое называется явлением конфигурации виджета. Затем при выборе виджета из списка, приведенного на главной странице для создания экземпляра виджета, Android активирует соответствующее явление конфигурации виджета. Это явление вам потребуется написать. Затем оно будет отвечать за конфигурацию экземпляра виджета.

При работе с показанным здесь Birthday Widget конфигурационное явление будет предупреждать вас, выводя имя именинника и дату наступающего дня рождения (рис. 13.4). Именно конфигурирующее явление должно сохранять эту информацию для долговременного использования и обновлять вид нужными значениями, устанавливаемыми конфигуратором.

ПРИМЕЧАНИЕ

Если пользователь решит создать на главном экране два экземпляра Birthday Widget, конфигурирующее явление будет вызвано дважды (по разу для каждого экземпляра виджета).

Внутри системы Android отслеживает количество экземпляров виджетов, присваивая им ID. Такие ID передаются обратным вызовам Java, а также конфигурирующему классу Java, так что обновления применяются именно к нужному

экземпляру. На рис. 13.3 во второй части строки Satya:3 число 3 — это ID виджета, точнее говоря, ID экземпляра виджета. Сам виджет идентифицируется по его имени компонента (одновременно являющемуся именем класса и пакета, к которому относится класс виджета). Понятия «ID виджета» и «ID экземпляра виджета» в данной главе полностью синонимичны. Чтобы проиллюстрировать этот момент, мы показали на рис. 13.3 экземпляр виджета.

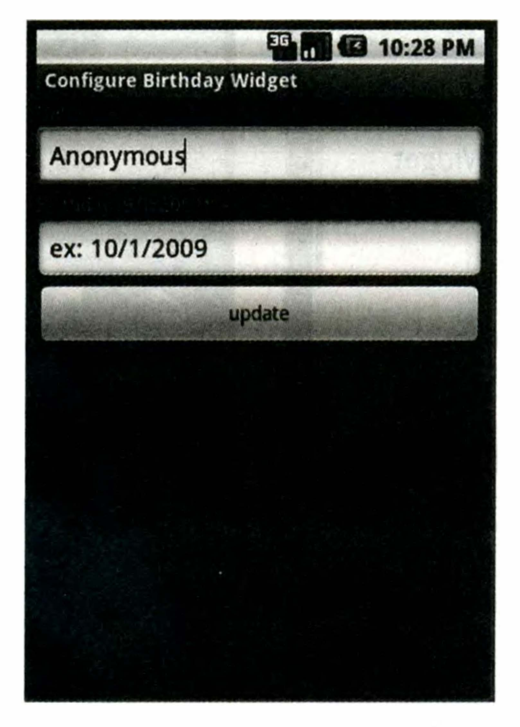


Рис. 13.4. Явление для конфигурации Birthday Widget

Рассмотрев виджет в общих чертах, более детально изучим его жизненный цикл.

Жизненный цикл виджета

Мы уже несколько раз упоминали об определении виджета. Кроме того, мы кратко упомянули о роли классов Java. В этом подразделе данные идеи будут изучены более детально, кроме того, будет рассмотрен жизненный цикл виджета. Он состоит из следующих этапов.

1. Определение виджета.
2. Создание экземпляра виджета.
3. `OnUpdate()` (при истечении заданного временного интервала).
4. Отклик на щелчки (при нажатии на виджет, расположенный на основном экране).

5. Удаление виджета (с основного экрана).
6. Деинсталляция.

Изучим эти этапы подробнее.

Определение виджета

Жизненный цикл виджета начинается с его определения. Определение приказывает Android отобразить имя виджета, активированного на основном экране, в списке виджетов (см. рис. 13.2). Для выполнения определения требуется два компонента. Во-первых, это класс Java, реализующий AppWidgetProvider, и вид-шаблон для виджета. Имея их, можно приступать к определению виджета в Android.

Определение виджета начинается с внесения следующей записи в файл описания Android, где указывается AppWidgetProvider (листинг 13.1).

Листинг 13.1. Определение виджета в файле описания Android

```
<manifest..>
<application>
....
   <receiver android:name=".BDayWidgetProvider">
      <meta-data android:name="android.appwidget.provider"
         android:resource="@xml/bday_appwidget_provider" />
      <intent-filter>
         <action android:name=
               "android.appwidget.action.APPWIDGET_UPDATE" />
      </intent-filter>
   </receiver>
   ...
   <activity>
      .....
   </activity>
<application>
</manifest>
```

В данном определении указан относящийся к широковещательному приемнику класс Java, называемый BDayWidgetProvider (как видите, он наследует свойства основного класса Android AppWidgetProvider из пакета виджета). Широковещательный приемник передает сообщения, касающиеся обновления виджетов программы.

ПРИМЕЧАНИЕ

Android передает сообщения об обновлениях в широковещательном режиме на основании частоты временного интервала.

В определении виджета в листинге 13.1 также содержится указание на XML-файл из каталога "/res/xml", который, в свою очередь, указывает вид виджета и частоту обновлений в соответствии с листингом 13.2.

Листинг 13.2. Определение вида виджета в XML-файле с информацией о поставщике виджетов

```
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   android:minWidth="150dp"
   android:minHeight="120dp"
```

```
    android:updatePeriodMillis="43200000"
    android:initialLayout="@layout/bday_widget"
    android:configure=
        "com.ai.android.BDayWidget.ConfigureBDayWidgetActivity"
    >
</appwidget-provider>
```

Этот XML-файл называется информационным файлом поставщика виджетов приложения (application widget provider information file). Внутри системы этот файл транслируется в класс Java AppWidgetProviderInfo. В этом файле указывается ширина и высота шаблона — соответственно 150 и 120 dp. Кроме того, в файле определения указывается частота обновлений — раз в 12 часов, — выраженная в миллисекундах. Это определение также указывает на файл шаблона (его мы разберем в листинге 13.7), описывающий внешний вид виджета.

Однако следует обратить внимание и на то, что шаблоны для таких видов могут содержать только элементы строго определенных типов. Комбинированный вид виджета относится к классу видов, называемому RemoteViews. Такие удаленные виды могут иметь лишь определенные типы дочерних видов. Допустимые составляющие подэлементов (subview) показаны в листинге 13.3.

Листинг 13.3. Элементы управления, которые могут присутствовать в RemoteViews

```
FrameLayout
LinearLayout
RelativeLayout

AnalogClock
Button
Chronometer
ImageButton
ImageView
ProgressBar
TextView
```

Этот список может отличаться в зависимости от версии Android. Основная причина, по которой существуют ограничения, связанные удаленными видами, заключается в том, что эти виды изолированы от процессов, которые ими управляют. Такие виды работают на базе определенного приложения, например Home. Управляющие элементы (контроллеры) этих видов — это фоновые процессы, активируемые таймерами. По этой причине такие виды и называются удаленными. Существует соответствующий класс Java RemoteViews, который предназначен для доступа к таким видам. Иными словами, программист не имеет прямого доступа к этим видам и не может применять к ним методы. Доступ к удаленным видам открывает только класс RemoteViews, играющий роль контроллера доступа (gatekeeper).

Все важные методы класса RemoteViews будут рассмотрены при изучении образца этого класса далее. Пока остановимся на том, что набор видов, которые могут присутствовать в файле шаблона виджета, жестко ограничен (см. листинг 13.3).

В определение виджета (см. листинг 13.2) также входит спецификация конфигурирующего явления, которое должно активироваться, когда пользователь созда-

ет экземпляр виджета. В листинге 13.2 это явление называется ConfigureBDayWidgetActivity. В таком явлении, как и в других явлениях Android, содержатся поля форм. Они используются для сбора информации, необходимой для экземпляра виджета.

Создание экземпляра виджета

Итак, все элементы XML, требуемые для определения виджета, уже готовы, а также доступны необходимые классы Java для работы с виджетами. Теперь рассмотрим, что происходит, когда пользователь выбирает один из виджетов в списке (см. рис. 13.2) для создания экземпляра виджета. Android активирует конфигурирующее явление (см. рис. 13.3), которое должно выполнить следующие задачи.

1. Получить ID экземпляра виджета от инициирующего намерения, запустившего конфигуратор.
2. Подсказать пользователю, что необходимо собрать специфичную информацию о виджете, для чего предоставляется нужный набор полей.
3. Сохранить информацию об экземпляре виджета так, чтобы при последующих вызовах update виджета имелся доступ к этой информации.
4. Подготовить вид виджета к первому отображению. Для этого нужно найти шаблон вида виджета и создать с ним объект RemoteViews.
5. Применить к объекту RemoteViews методы, необходимые для установки значений, которые описывают конкретные элементы вида — текст, изображение и т. д.
6. Использовать объект RemoteViews для регистрации всех событий onClick при щелчках на любых подэлементах виджета.
7. Приказать AppWidgetManager нарисовать RemoteViews на основном экране, используя ID экземпляра этого виджета.
8. Вернуть ID виджета и закрыться.

После возврата этого явления Android нарисует вид виджета в соответствии с указаниями конфигуратора. Обратите внимание: первую операцию рисования в данном случае выполняет именно конфигуратор, а не метод onUpdate(), относящийся к AppWidgetProvider.

ПРИМЕЧАНИЕ

Конфигурирующее явление использовать не обязательно. Если такое явление не задано, вызов направляется непосредственно к onUpdate(), в том числе в первый раз. Тогда onUpdate() отвечает за обновление вида.

Android повторяет этот процесс с каждым экземпляром виджета, который создает пользователь. Обратите также внимание на то, что в документации отдельно не указано, что пользователь может одновременно работать только с одним виджетом.

Наряду с активацией конфигурирующего явления, Android активирует обратный вызов onEnabled, относящийся к классу AppWidgetProvider. Кратко рассмотрим обратные вызовы класса AppWidgetProvider, для чего изучим оболочку нашего BDayWidgetProvider (листинг 13.4). Весь код этого файла будет приведен позднее, в листинге 13.9.

Листинг 13.4. Оболочка поставщика виджетов

```
public class BDayWidgetProvider extends AppWidgetProvider
{
   public void onUpdate(Context context,
                 AppWidgetManager appWidgetManager,
                 int[] appWidgetIds){}

   public void onDeleted(Context context, int[] appWidgetIds){}
   public void onEnabled(Context context){}
   public void onDisabled(Context context) {}
}
```

Метод обратного вызова onEnabled() указывает, что имеется как минимум один экземпляр виджета, работающий на основном экране. Это означает, что пользователь создавал на домашней странице виджет как минимум один раз. В ходе данного вызова необходимо обеспечить доставку сообщений к этому компоненту (подробно этот механизм будет показан в листинге 13.9). В Android классы иногда называются компонентами, особенно такие классы, которые могут использоваться многократно, например Activity, Service или BroadcastReceiver. В таком случае базовый класс AppWidgetProvider выполняет функцию широковещательного приемника. Ему можно включить или отключить получение широковещательных сообщений.

Метод обратного вызова onDeleted() задействуется в тех случаях, когда пользователь перемещает экземпляр виджета в корзину. На этом этапе следует удалять все постоянные значения (persistent values), сохраненные для данного экземпляра виджета.

Метод обратного вызова onDisabled() задействуется после удаления последнего экземпляра виджета с основного экрана. Это происходит при переносе последнего экземпляра виджета в корзину. Данный метод используется для того, чтобы отменить получение любых широковещательных сообщений, направляемых к этому компоненту (это подробнее показано в листинге 13.9).

Метод обратного вызова onUpdate() задействуется всякий раз по истечении времени, которое задано для таймера, указанного в листинге 13.2. Кроме того, данный метод вызывается в самый первый раз, когда экземпляр виджета создается, а конфигурирующее явление при этом отсутствует. Если конфигурирующее явление применяется, то при создании экземпляра виджета этот метод не вызывается. Впоследствии этот метод вызывается всякий раз по истечении времени, заданного на таймере.

OnUpdate

Когда экземпляр виджета уже есть на основном экране, следующее важное событие — это истечение времени, заданного на таймере. Как было указано выше, в ответ на истечение времени таймера Android вызовет метод onUpdate() посредством широковещательного приемника. Это означает, что загружается соответствующий процесс Java, в рамках которого определяется onUpdate(), и этот процесс продолжает действовать вплоть до завершения вызова. После возращения результата вызова процесс можно завершить.

Кроме того, если на обработку полученного ответа требуется значительное время, рекомендуется запускать локальную службу, которая будет выполнять эту работу. Таким образом, вы обеспечиваете возвращение широковещательного потока. Служба будет работать в отдельном потоке, специально выделенном для выполнения этой служебной задачи.

Так или иначе, когда в методе onUpdate() появятся доступные данные, вы можете вызвать AppWidgetManager для рисования удаленного вида, который нужно будет обновить при помощи этих данных. Подразумевается, что если бы вместо этого для обновления активировалась специальная служба, то ID виджета нужно было бы сообщить как дополнительную информацию тому намерению, которое запускает службу.

Это подтверждает, что класс AppWidgetProvider не поддерживает состояний (stateless) и, вероятно, даже не сможет сохранять статические переменные между активациями. Объяснить этот феномен можно тем, что процесс Java, содержащий этот класс широковещательного приемника, может быть удален и реконструирован между двумя активациями, в результате чего произойдет повторная инициализация статических переменных.

В результате нам может понадобиться схема для запоминания состояний. Если обновления происходят не очень часто (например, каждые несколько секунд), весьма целесообразно будет сохранять состояние экземпляра виджета в постоянном хранилище (persistent store), например в файле совместно используемых настроек либо в базе данных SQLLite. В следующем примере мы применим совместно используемые настройки в качестве API для долгосрочного хранения данных.

ВНИМАНИЕ

В целях экономии энергии в Android настоятельно рекомендуется задавать для обновлений длительность более часа, чтобы устройство не приходилось активировать слишком часто. Кроме того, в будущих версиях может быть жестко задана минимальная длительность обновления — 30 минут или более.

Если обновление занимает гораздо меньше времени, например всего несколько секунд, то вы должны вызывать этот метод onUpdate() сами при помощи функций класса AlarmManager. При использовании AlarmManager можно также не вызывать onUpdate(), а применять его при экстренных обратных вызовах (alarm callbacks).

Метод onUpdate() обычно используется для решения следующих задач.

- Убедиться, что конфигуратор завершил работу, в противном случае — просто произвести возврат. Это не должно составлять проблем в версиях 2.0 и выше, где длительность обновлений, как ожидается, будет увеличена. Возможен случай, при котором onUpdate() будет вызываться еще до завершения работы конфигуратора.
- Находить сохраненные данные для конкретного экземпляра виджета.
- Находить шаблон вида виджета и создавать с ним объект RemoteViews.
- Применять методы к RemoteViews, чтобы задавать значения для отдельных элементов вида, например для текста, изображений и т. д.
- Регистрировать любые события onClick (то есть щелчки) с любыми видами, применяя для этого отложенные намерения (pending intents).
- Приказывать AppWidgetManager рисовать RemoteViews, используя ID экземпляра.

Как видите, задачи и функции конфигуратора и метода onUpdate() во многом совпадают. Возможно, вы пожелаете использовать с ними обоими одинаковые функции.

Выполнение обратных вызовов в ответ на щелчки на виджете

Как было указано выше, метод onUpdate() поддерживает актуальность видов виджета. Вид виджета, а также входящие в его состав подэлементы могут выполнять обратные вызовы, если на них сделан щелчок (указателем мыши). Обычно с методом onUpdate() применяются отложенные намерения, регистрирующие действие для реакции на событие, например на щелчок указателем мыши. Такое действие затем может запускать службу или явление, скажем, открывать браузер.

Затем активированная таким образом служба или явление могут выполнить обратный вызов вида, используя при необходимости ID экземпляра виджета и AppWidgetManager. По этой причине важно, чтобы вместе с отложенным намерением передавался ID экземпляра виджета.

Удаление экземпляра виджета

Еще одно важное событие, которое может произойти с виджетом, — его удаление. Для этого пользователь должен щелкнуть на виджете, расположенном на экране. Таким образом активируется корзина, отображаемая в нижней части домашней страницы. После этого виджет можно будет перетащить в корзину.

При этом также вызывается метод onDelete() поставщика виджетов. Если вы сохранили какую-либо информацию о состоянии конкретного экземпляра виджета, эти данные удаляются при помощи того же метода onDelete.

Кроме того, если только что удаленный виджет был последним экземпляром виджетов определенного типа, Android вызывает метод onDisable(). Данный обратный вызов применяется для удаления всех долговременных атрибутов, сохраненных для всех экземпляров виджета, и отмены регистрации обратных вызовов метода onUpdate(), направляемых к данному виджету (см. листинг 13.9).

Деинсталляция пакетов виджетов

Итак, мы полностью изучили жизненный цикл виджета. Перед тем как перейти к следующему разделу, кратко упомянем о том, что, собираясь деинсталлировать или установить новую версию файла APK, содержащего виджеты, нужно удалять старые виджеты.

Перед деинсталляцией пакета рекомендуется удалить все экземпляры виджетов. Следуйте указаниям, данным в разделе «Удаление экземпляра виджета», и правильно удалите все виджеты.

Затем можно деинсталлировать имеющийся релиз пакета и установить новый. Это особенно важно делать при разработке виджетов с применением Eclipse ADT, поскольку при работе ADT пытается сделать это всякий раз, когда вы запускаете эту программу. Итак, между запусками необходимо убедиться, что все экземпляры виджетов удалены.

Пример приложения, работающего с виджетами

Мы уже изучили теоретические и практические аспекты виджетов. Используем эти знания и создадим образец виджета, который мы разработаем, протестируем и применим на практике.

Цель следующего упражнения — создать виджет, который будет напоминать о наступлении дней рождения. В каждом экземпляре виджета будет отображаться имя, дата следующего дня рождения и количество дней, оставшихся от сегодняшнего дня до этой даты. Кроме того, мы создадим область `onClick`, на которой можно будет щелкать, чтобы покупать подарки. При щелчке будет открываться браузер и загружаться сайт http://www.google.com.

Готовый виджет должен выглядеть как на рис. 13.5.



Рис. 13.5. Внешний вид виджета, напоминающего о наступлении дней рождения

Для реализации этого виджета нужны следующие файлы. В зависимости от того, каким исходным пакетом Java вы будете пользоваться, файлы Java будут находиться в подкаталоге `src`. За ним последует дерево каталогов, в которых вы размещаете пакеты Java. Для краткости и экономии места мы использовали вместо подкаталогов многоточие (...).

- `AndroidManifest.xml //` — здесь определяется `AppWidgetProvider`.
- `res/xml/bday_appwidget_provider.xml //` — параметры и расположение виджетов.
- `res/layout/bday_widget.xml //` — шаблон виджета.
- `res/drawable/box1.xml //` — здесь предоставляются рамки для частей шаблона виджета.
- `src/.../BDayWidgetProvider //` — реализация класса `AppWidgetProvider`.

При внедрении виджета также используются следующие файлы, предназначенные для управления состоянием виджета:

- `src/.../IWidgetModelSaveContract //` — контракт для сохранения модели виджета;
- `src/.../APrefWidgetModel //` — абстрактная модель виджета, работа которой основана на настройках;
- `src/.../BDayWidgetModel //` — модель виджета, в которой хранятся данные для его вида;
- `src/.../Utils.java //` — несколько вспомогательных классов.

Кроме того, при внедрении виджета используются и следующие файлы, отвечающие за конфигурирующее явление:

- src/.../ConfigureBDayWidgetActivity.java // — конфигурирующее явление;
- layout/edit_bday_widget.xml // — шаблон для взятия информации об имени и дне рождения.

Мы рассмотрим все эти файлы и объясним все дополнительные феномены, о которых пока не упомянули. По завершении данного раздела вы сможете скопировать эти файлы и вставить их в свой проект, чтобы можно было создать и протестировать виджет в собственной рабочей среде.

Определение поставщика виджета

Определение виджета начинается в файле описания приложения Android. Именно здесь задаются поставщик виджета, конфигурирующее явление виджета и указатель на другой XML-файл, где, в свою очередь, определяется шаблон виджета.

Все эти элементы, используемые в виджете, который напоминает о наступлении дней рождения, выделены полужирным шрифтом в следующем файле описания Android (листинг 13.5). Обратите внимание на определение BDayAppWidgetProvider, представляющего собой широковещательный приемник, а также на определение конфигурирующего явления ConfigureBDayWidgetActivity.

Листинг 13.5. Файл описания Android для приложения-образца BDayWidget

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.ai.android.BDayWidget"
   android:versionCode="1"
   android:versionName="1.0.0">
   <application android:icon=
           "@drawable/icon" android:label="Birthday Widget">
<!--
**************************************************************************
* Приемник для поставщика виджета, напоминающего о днях рождения.
**************************************************************************
-->
   <receiver android:name=".BDayWidgetProvider">
      <meta-data android:name="android.appwidget.provider"
         android:resource="@xml/bday_appwidget_provider" />
      <intent-filter>
         <action android:name=
              "android.appwidget.action.APPWIDGET_UPDATE" />
      </intent-filter>
   </receiver>
<!--
**************************************************************************
* Конфигурирующее явление поставщика виджета, напоминающего о днях рождения.
**************************************************************************
-->
   <activity android:name=".ConfigureBDayWidgetActivity"
           android:label="Configure Birthday Widget">
```

```
      <intent-filter>
         <action android:name=
              "android.appwidget.action.APPWIDGET_CONFIGURE" />
      </intent-filter>
   </activity>

   </application>
   <uses-sdk android:minSdkVersion="3" />
</manifest>
```

ПРИМЕЧАНИЕ

Принимающий узел является смежным по отношению к узлу явления. Это указано в файле описания. Кроме того, принимающий узел является непосредственным дочерним элементом узла приложения.

Метка приложения, идентифицируемого по "Birthday Widget" в следующей строке:

```
<application android:icon="@drawable/icon" android:label="Birthday Widget ">
```

также отображается в списке (см. рис. 13.2) на домашней странице. Если вы создаете определение виджета впервые, обязательно точно воспроизведите следующую строку:

```
<meta-data android:name="android.appwidget.provider"
```

Спецификация "android.appwidget.provider" используется только в Android и должна быть отмечена при помощи следующих строк:

```
<intent-filter>
   <action android:name="android.appwidget.action.APPWIDGET_UPDATE" />
</intent-filter>
```

Наконец, определение конфигурирующего явления не отличается от определения любого другого явления, за исключением того, что конфигурирующее явление необходимо объявить как способное реагировать на действия APPWIDGET_CONFIGURE.

Определение размера виджета

Хотя в файле описания Android и определяется поставщик виджетов, дополнительная информация о виджете содержится в отдельном XML-файле. К дополнительной информации относятся размер виджета, название файла шаблона виджета, период, через который происходят обновления, и название компонента конфигурирующего явления (или соответствующего класса).

Дополнительный XML-файл, идентифицируется по узлу android:resource из предыдущего определения поставщика виджета (см. листинг 13.5). В листинге 13.6 показан файл с информацией об этом поставщике (/res/xml/bday_appwidget_provider.xml).

Листинг 13.6. Определение вида виджета для BDayWidget

```
<!-- res/xml/bday_appwidget_provider.xml -->
<appwidget-provider xmlns:android="http://schemas.android.com/apk/res/android"
   android:minWidth="150dp"
   android:minHeight="120dp"
```

```
    android:updatePeriodMillis="4320000"
    android:initialLayout="@layout/bday_widget"
    android:configure=
          "com.ai.android.BDayWidget.ConfigureBDayWidgetActivity"
    >
</appwidget-provider>
```

В этом файле системе Android указывается нужная ширина и высота в пикселах. Однако Android округляет эти величины до целого количества ячеек. Дело в том, что ОС Android организует площадь основного экрана устройства в виде матрицы из ячеек. Каждая ячейка имеет высоту и ширину 74 пиксела, не зависящие от плотности экрана (dp). В Android рекомендуется задавать высоту и ширину в числах, кратных количеству ячеек минус 2 пиксела (для поправки на округление и т. д.).

Кроме того, в этом файле также указывается, как часто следует вызывать метод `onUpdate()`. В Android настоятельно рекомендуется выполнять обновления не чаще нескольких раз в день. Чтобы обновления не производились, здесь можно поставить значение `0`. Это целесообразно, если вы собираетесь сами управлять обновлениями виджета при помощи класса `Alarm Manager`.

Атрибут первичного шаблона (initial layout attribute) указывает на используемый в данный момент шаблон виджета (подробнее это приведено в листинге 13.7). Наконец, атрибут конфигурации указывает на класс конфигурирующего явления. При определении этот класс должен быть полностью квалифицирован.

Теперь изучим шаблон виджета, которым будем пользоваться.

Файлы, относящиеся к шаблону виджета

В предыдущем разделе и листинге 13.6 было показано, что шаблон виджета определяется в специальном файле шаблона. Этот файл шаблона очень похож на аналогичные файлы любых других видов, используемых в Android.

Однако, чтобы как-то стандартизировать работу с виджетами, в документации Android есть рекомендации по разработке виджетов. Они приводятся по адресу **http://developer.android.com/guide/practices/ui_guidelines/widget_design.html**.

Кроме рекомендаций, на этом ресурсе предлагается набор фонов, при помощи которых вы можете улучшить внешний вид своих виджетов. В нашем примере мы воспользуемся немного иным подходом и применим обычные шаблоны видов с контурными фонами.

Файл шаблона виджета

В листинге 13.7 представлен файл шаблона, который мы использовали при создании виджета, показанного на рис. 13.5.

Листинг 13.7. Определение шаблона вида виджета для BDayWidget

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="150dp"
```

```
    android:layout_height="120dp"
    android:background="@drawable/box1"
    >
<TextView
    android:id="@+id/bdw_w_name"
    android:layout_width="fill_parent"
    android:layout_height="30dp"
    android:text="Anonymous"
    android:background="@drawable/box1"
    android:gravity="center"
    />
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="60dp"
    >
    <TextView
        android:id="@+id/bdw_w_days"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:text="0"
        android:gravity="center"
        android:textSize="30sp"
        android:layout_weight="50"
        />
    <TextView
        android:id="@+id/bdw_w_button_buy"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:textSize="20sp"
        android:text="Buy"
        android:layout_weight="50"
        android:background="#FF6633"
        android:gravity="center"
    />
</LinearLayout>
<TextView
    android:id="@+id/bdw_w_date"
    android:layout_width="fill_parent"
    android:layout_height="30dp"
    android:text="1/1/2000"
    android:background="@drawable/box1"
    android:gravity="center"
    />
</LinearLayout>
```

В данном шаблоне для получения желаемого эффекта используются вложенные узлы LinearLayout. Некоторые элементы также применяют файл определения контура box1.xml — здесь определяются границы вида.

Файл фонового контура вида

Код для определения этого контура показан в листинге 13.8 (файл должен находиться в подкаталоге /res/drawable).

Листинг 13.8. Определение контура описывающего прямоугольника

```
<!-- res/drawable/box1.xml -->
<shape xmlns:android="http://schemas.android.com/apk/res/android">
   <stroke android:width="4dp" android:color="#888888" />
   <padding android:left="2dp" android:top="2dp"
         android:right="2dp" android:bottom="2dp" />
   <corners android:radius="4dp" />
</shape>
```

Мы создали шаблон именно таким образом, поскольку подобный шаблон можно использовать не только с виджетами, но и с другими элементами. Возможно, вы захотите создать явление и протестировать шаблоны отдельно, прежде чем опробовать их вместе с виджетом (мы, например, сделали именно так). Мы не сразу получили виджет, который нам понравился. Эксперименты с готовыми виджетами могут быть довольно утомительны: перед каждым перезапуском программы нужно будет удалить виджеты, деинсталлировать, снова инсталлировать, а потом опять перетащить виджеты на домашнюю страницу.

Мы уже обсудили все файлы, необходимые для определения виджета в формате XML. Теперь рассмотрим, как реагировать на события, связанные с жизненным циклом виджета. Для этого следует изучить класс поставщика виджетов.

Реализация поставщика виджетов

Рассуждая об архитектуре виджетов, мы упоминали о функциях класса поставщика виджетов. В поставщике виджетов должны быть реализованы следующие методы обратного вызова широковещательных приемников:

- onUpdate();
- onDelete();
- onEnable();
- onDisable().

Код Java в листинге 13.9 демонстрирует реализацию каждого из этих методов.

Листинг 13.9. Образец реализации поставщика виджетов: BDayWidgetProvider

```
///src/<your-package>/BDayWidgetProvider.java
public class BDayWidgetProvider extends AppWidgetProvider
{
   private static final String tag = "BDayWidgetProvider";
   public void onUpdate(Context context,
                  AppWidgetManager appWidgetManager,
                  int[] appWidgetIds) {
      final int N = appWidgetIds.length;
      for (int i=0; i<N; i++)
      {
```

```
            int appWidgetId = appWidgetIds[i];
            updateAppWidget(context, appWidgetManager, appWidgetId);
        }
    }

    public void onDeleted(Context context, int[] appWidgetIds) {
        final int N = appWidgetIds.length;
        for (int i=0; i<N; i++) {
            BDayWidgetModel.removePrefs(context, appWidgetIds[i]);
        }
    }
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();
        if (AppWidgetManager.ACTION_APPWIDGET_DELETED.equals(action)) {
            Bundle extras = intent.getExtras();
            final int appWidgetId = extras.getInt
                        (AppWidgetManager.EXTRA_APPWIDGET_ID,
                        AppWidgetManager.INVALID_APPWIDGET_ID);

            if (appWidgetId != AppWidgetManager.INVALID_APPWIDGET_ID) {
                this.onDeleted(context, new int[] { appWidgetId });
            }
        }
        else {
            super.onReceive(context, intent);
        }
    }

    public void onEnabled(Context context) {
        BDayWidgetModel.clearAllPreferences(context);
        PackageManager pm = context.getPackageManager();
        pm.setComponentEnabledSetting(
            new ComponentName("com.ai.android.BDayWidget",
                        ".BDayWidgetProvider"),
            PackageManager.COMPONENT_ENABLED_STATE_ENABLED,
            PackageManager.DONT_KILL_APP);
    }

    public void onDisabled(Context context) {
        BDayWidgetModel.clearAllPreferences(context);
        PackageManager pm = context.getPackageManager();
        pm.setComponentEnabledSetting(
            new ComponentName("com.ai.android.BDayWidget",
                        ".BDayWidgetProvider"),
            PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
            PackageManager.DONT_KILL_APP);
    }

    private void updateAppWidget(Context context,
                        AppWidgetManager appWidgetManager,
```

```
                    int appWidgetId) {
        BDayWidgetModel bwm =
                BDayWidgetModel.retrieveModel(context, appWidgetId);
        if (bwm == null) {
            return;
        }
        ConfigureBDayWidgetActivity
                .updateAppWidget(context, appWidgetManager, bwm);
    }
}
```

Чтобы вспомнить, какие задачи выполняются в каждом из этих методов, обратитесь к разделу «Архитектура виджетов основного экрана». В виджете, напоминающем о наступлении дней рождения, все эти методы по очереди используют методы класса BDayWidgetModel. В числе этих методов — removePrefs(), retrievePrefs() и clearAllPreferences().

Класс BDayWidgetModel применяется для содержания состояний экземпляров нашего виджета (данный класс будет рассмотрен далее). Чтобы разобраться с классом этого поставщика виджетов, нам всего лишь нужно знать, что в данном случае для получения данных, необходимых для этого экземпляра виджета, применяется класс моделей (model class). Эти данные сохраняются в настройках. Именно поэтому методы и называются removePrefs(), retrievePrefs() и clearAllPreferences() (названия станут понятнее, если заменить в них Prefs на Data — чтобы получилось removeData(), retrieveData() и clearAllData()). В любом случае такое преобразование сделано лишь для того, чтобы подчеркнуть природу методов, на практике вы не встретите подобных названий с суффиксом Data.

Как было указано выше, обновляющий метод вызывается для всех экземпляров виджета. Из ID экземпляров виджетов создается массив, который затем используется при передаче. К каждому ID применяется метод onUpdate(), находящий модель соответствующего экземпляра виджета и вызывающий тот же метод, что используется конфигурирующим явлением (см. листинг 13.14). Это делается для отображения найденной модели виджета.

В методе onDelete() мы инстанцировали BDayWidgetModel, а потом приказали ему удалить самого себя из постоянного хранилища настроек.

В методе onEnabled(), который вызывается всего один раз (когда задействуется первый экземпляр), мы удалили все сохраненные в долговременном хранилище модели виджетов и начали, так сказать, с чистого листа. То же самое мы делаем с методом onDisabled(), так что от образцов виджетов не остается и следа.

В методе onEnabled() активируется компонент поставщика виджетов так, чтобы виджет мог принимать широковещательные сообщения. В методе onDisabled() этот компонент деактивируется, в результате виджет больше не будет искать широковещательных сообщений.

ПРИМЕЧАНИЕ

Особый случай представляет собой метод onReceive(). До версии 1.6 в системе присутствовал баг, из-за которого метод onDelete() не вызывался. Для решения этой проблемы в Android прямо предоставляется метод onReceive(). В версии 1.6 и выше этот метод больше не нужен; вместо него применяется такой же метод из базового класса.

При реализации идеи моделей виджетов код остается чистым. Далее мы рассмотрим модели виджетов и их реализацию.

Реализация моделей виджетов

Что такое модель виджета? Эта концепция встречается не только в Android. Однако если вы знакомы с традиционным программированием пользовательских интерфейсов, то, вероятно, знаете о концепции MVC (Model – View – Controller, модель – представление – контроллер). Здесь модель содержит данные, необходимые для вида. Вид отвечает за отображение, а контроллер играет опосредующую роль между видом и моделью.

Хотя в SDK Android для этого не требуется применять специфический подход, мы воспользовались этой идеей, чтобы упростить программирование виджетов. При использовании такого подхода для вида каждого экземпляра виджета создается эквивалентный класс Java, называемый моделью виджета. Эта модель содержит все методы, которые могут передавать необходимые данные экземплярам виджета.

Кроме механизма подачи данных, мы создаем для этих моделей несколько базовых классов, чтобы модели могли сами себя сохранять в долговременном хранилище данных (например, в совместно используемых настройках) и извлекать себя оттуда. Мы изучим иерархическую структуру класса моделей и покажем, как при помощи совместно используемых настроек сохранять и получать данные.

Интерфейс для модели виджета

Сначала поговорим об интерфейсе, действующем в качестве контракта для модели виджета – с его помощью модель виджета может объявлять, какие поля должны сохраняться в долговременном хранилище данных. Кроме того, в контракте определяется, как установить поле при получении его из базы данных и до передачи запрашивающему клиенту.

В листинге 13.10 показан исходный код этого интерфейса.

Листинг 13.10. Сохранение состояния виджета: контракт

```
// имя файла: src/…/IWidgetModelSaveContract.java
public interface IWidgetModelSaveContract
{
   public void setValueForPref(String key, String value);
   public String getPrefname();

   // возврат пар «ключ – значение», которые требуется сохранить
   public Map<String,String> getPrefsToSave();

   // вызывается после восстановления
   public void init();
}
```

Этот интерфейс разработан таким образом, что в производном абстрактном классе осуществляется реализация с применением специального долговременного

хранилища данных. Как было указано выше, для долговременного хранения данных мы воспользуемся совместно используемыми настройками, применяемыми в Android. Из названия этого интерфейса понятно, что он выполняет только функцию сохранения. Клиенты, например BDayWidgetProvider, по-прежнему будут работать в данном интерфейсе на основе того полного (most-derived) класса, который предназначен для выполнения конкретных методов.

ПРИМЕЧАНИЕ

В реальных программах такое наследование будет структурироваться немного иначе: вероятно, будет применяться не наследование как таковое, а механизм делегирования для повторного использования. Однако данная иерархия наследования отлично подходит для нашего теста, чтобы продемонстрировать работу с моделями виджетов.

Теперь рассмотрим абстрактную реализацию, в которой поля данных виджета сохраняются как совместно используемые настройки.

Абстрактная реализация модели виджета

Весь код, отвечающий за взаимодействие с долговременным хранилищем данных, реализуется в классе APrefWidgetModel. Pref в названии этого класса соответствует Preference (настройка), поскольку в этом классе для сохранения данных о модели виджета используется функция Android SharedPrferences.

Кроме того, в данном классе представлена идея базового виджета. В поле iid отображается ID экземпляра виджета. Для этого класса всегда нужен конструктор, принимающий ID экземпляра виджета в качестве аргумента для выполнения требования, связанного с ID экземпляра.

В листинге 13.11 приведен исходный код этого класса. Основные методы класса выделены полужирным шрифтом.

Листинг 13.11. Реализация сохранения виджета при помощи совместно используемых настроек

```
// имя файла: /src/…/APrefWidgetModel.java
public abstract class APrefWidgetModel
implements IWidgetModelSaveContract
{
   private static String tag = "AWidgetModel";

   public int iid;
   public APrefWidgetModel(int instanceId) {
      iid = instanceId;
   }
   // абстрактные методы
   public abstract String getPrefname();
   public abstract void init();
   public Map<String,String> getPrefsToSave(){ return null;}

   public void savePreferences(Context context){
      Map<String,String> keyValuePairs = getPrefsToSave();
      if (keyValuePairs == null){
         return;
```

```
        }
        // здесь мы сохраним некоторые значения
        SharedPreferences.Editor prefs =
            context.getSharedPreferences(getPrefname(), 0).edit();

        for(String key: keyValuePairs.keySet()){
            String value = keyValuePairs.get(key);
            savePref(prefs,key,value);
        }
        // итоговая передача значений
        prefs.commit();
    }

    private void savePref(SharedPreferences.Editor prefs,
                    String key, String value) {
        String newkey = getStoredKeyForFieldName(key);
        prefs.putString(newkey, value);
    }
    private void removePref(SharedPreferences.Editor prefs, String key) {
        String newkey = getStoredKeyForFieldName(key);
        prefs.remove(newkey);
    }
    protected String getStoredKeyForFieldName(String fieldName){
        return fieldName + "_" + iid;
    }
    public static void clearAllPreferences(Context context,
                                        String prefname) {
        SharedPreferences prefs=context.getSharedPreferences(prefname, 0);
        SharedPreferences.Editor prefsEdit = prefs.edit();
        prefsEdit.clear();
        prefsEdit.commit();
    }

    public boolean retrievePrefs(Context ctx) {
        SharedPreferences prefs =
                ctx.getSharedPreferences(getPrefname(), 0);
        Map<String,?> keyValuePairs = prefs.getAll();
        boolean prefFound = false;
        for (String key: keyValuePairs.keySet()){
            if (isItMyPref(key) == true){
                String value = (String)keyValuePairs.get(key);
                setValueForPref(key,value);
                prefFound = true;
            }
        }
        return prefFound;
    }

    public void removePrefs(Context context) {
        Map<String,String> keyValuePairs = getPrefsToSave();
        if (keyValuePairs == null){
```

```
            return;
        }
        // здесь мы сохраним некоторые значения
        SharedPreferences.Editor prefs =
            context.getSharedPreferences(getPrefname(), 0).edit();

        for(String key: keyValuePairs.keySet()){
            removePref(prefs,key);
        }
        // итоговая передача значений
        prefs.commit();
    }
    private boolean isItMyPref(String keyname) {
        if (keyname.indexOf("_" + iid) > 0){
            return true;
        }
        return false;
    }
    public void setValueForPref(String key, String value) {
        return;
    }
}
```

Посмотрим, как реализуются основные методы этого класса. Начнем с сохранения атрибутов модели виджета в файле совместно используемых настроек:

```
public void savePreferences(Context context)
{
    Map<String,String> keyValuePairs = getPrefsToSave();
    if (keyValuePairs == null){ return; }

    // здесь мы сохраним некоторые значения
    SharedPreferences.Editor prefs =
    context.getSharedPreferences(getPrefname(), 0).edit();

    for(String key: keyValuePairs.keySet()){
        String value = keyValuePairs.get(key);
            savePref(prefs,key,value);
    }
    // итоговая передача значений
    prefs.commit();
}
```

Этот метод начинает работу, приказывая производному классу вернуть таблицу соответствия ключей и значений, где ключи — это атрибуты модели, а значения — это строковые представления (string representations) данных значений атрибутов. Затем прикажем Android `context` сохранить файл `SharedPreferences` посредством `context.getSharedPreferences()`. Данный API требует задать уникальное имя пакета. Это имя предоставляется производной моделью.

Когда мы получим совместно используемые настройки, следуя указаниям из документации Android, запросим у системы редактируемую версию совместно ис-

пользуемых настроек. Затем мы по очереди обновим настройки. После завершения применим к настройкам метод commit(), чтобы они сохранились в долговременной памяти.

Чтобы лучше разобраться в деталях, изучите ссылки API, касающиеся классов SharedPreferences и SharedPreferences.Editor. В разделе этой главы, посвященном дополнительным ресурсам, есть соответствующие URL. Кроме того, необходимо отметить, что файлы совместно используемых настроек относятся к типу XML и их можно найти в каталоге с данными соответствующего пакета.

Поскольку мы сохраняли все данные об экземплярах виджетов в одном и том же файле, нам нужен способ для различения одноименных полей, относящихся к различным экземплярам виджетов. Например, если у нас будет два экземпляра виджетов, с именами 1 и 2, то нам понадобится два ключа для сохранения атрибута Name. Таким образом, имеем name_1 и name_2. Это преобразование осуществляется так:

```
protected String getStoredKeyForFieldName(String fieldName) {
   return fieldName + "_" + iid;
}
```

В производном классе данный метод также используется, для проверки того, какое поле обновлять при вызове метода setValue().

Реализация модели виджета, используемой с Birthday Widget

В конечном счете полный класс в данной иерархии моделей виджетов отвечает за поддержку работы со всеми полями, которые требуются конкретному виду. При операциях сохранения и нахождения информации он опирается на базовые классы. Мы создали этот полный класс таким образом, что все клиенты, работающие с этими моделями, взаимодействуют напрямую с полным классом, поскольку этот класс для них является наиболее подходящим.

Например, когда экземпляр виджета впервые создается конфигурирующим явлением, это явление инстанцирует один из указанных классов, заносит значения и приказывает сохранить себя.

В связи с требованиями, касающимися данного конкретного вида, этот класс обеспечивает работу с тремя полями:

- name — имя человека;
- bday — дата следующего дня рождения;
- url — гиперссылка, по которой следует перейти для покупки подарков.

Затем в классе следует вычисленный атрибут под названием howManyDays, представляющий собой количество дней от сегодняшнего до даты следующего дня рождения.

Кроме того, как видите, этот класс отвечает за выполнение контракта сохранения. При этом используются следующие методы:

```
public void setValueForPref(String key, String value);
public String getPrefname();
public Map<String,String> getPrefsToSave();
```

В листинге 13.12 представлен код, организующий взаимодействие всех этих компонентов.

Листинг 13.12. BDayWidgetModel: реализация модели состояния

```
// имя файла: /src/…/BDayWidgetModel.java
public class BDayWidgetModel extends APrefWidgetModel
{
   private static String tag="BDayWidgetModel";

   // предоставление уникального имени для сохранения даты
   private static String BDAY_WIDGET_PROVIDER_NAME=
      "com.ai.android.BDayWidget.BDayWidgetProvider";

   // переменные для рисования вида виджета
   private String name = "anon";
   private static String F_NAME = "name";

   private String bday = "1/1/2001";
   private static String F_BDAY = "bday";

   private String url="http://www.google.com";

   // конструктор/получение/установка
   public BDayWidgetModel(int instanceId){
      super(instanceId);
   }
   public BDayWidgetModel(int instanceId, String inName, String inBday){
      super(instanceId);
      name=inName;
      bday=inBday;
   }
      public void init(){}
      public void setName(String inname){name=inname;}
      public void setBday(String inbday){bday=inbday;}

      public String getName(){return name;}
      public String getBday(){return bday;}

      public long howManyDays(){
         try {
            return Utils.howfarInDays(Utils.getDate(this.bday));
         }
         catch(ParseException x){
            return 20000;
         }
      }

   // реализация контракта сохранения

   public void setValueForPref(String key, String value){
      if (key.equals(getStoredKeyForFieldName(BDayWidgetModel.F_NAME))){
```

```
            this.name = value;
            return;
        }
        if (key.equals(getStoredKeyForFieldName(BDayWidgetModel.F_BDAY))){
            this.bday = value;
            return;
        }
    }

    public String getPrefname() {
        return BDayWidgetModel.BDAY_WIDGET_PROVIDER_NAME;
    }

    // возвращение пар ключ – значение, которые требуется сохранить
    public Map getPrefsToSave() {
        Map map
        = new HashMap();
        map.put(BDayWidgetModel.F_NAME, this.name);
        map.put(BDayWidgetModel.F_BDAY, this.bday);
        return map;
    }
    public String toString() {
        StringBuffer sbuf = new StringBuffer();
        sbuf.append("iid:" + iid);
        sbuf.append("name:" + name);
        sbuf.append("bday:" + bday);
        return sbuf.toString();
    }
    public static void clearAllPreferences(Context ctx){
        APrefWidgetModel.clearAllPreferences(ctx,
                BDayWidgetModel.BDAY_WIDGET_PROVIDER_NAME);
    }

    public static BDayWidgetModel retrieveModel(Context ctx, int widgetId){
        BDayWidgetModel m = new BDayWidgetModel(widgetId);
        boolean found = m.retrievePrefs(ctx);
        return found ? m:null;
    }
}
```

Как видите, в этом классе используется пара утилит, предназначенных для хранения дат. Прежде чем перейти к обсуждению реализации конфигурирующих явлений виджетов, мы приведем исходный код этих утилит.

Несколько утилит для работы с датой

Далее показан вспомогательный класс, используемый для работы с датами. Он принимает строку с данными и проверяет, является ли она валидной датой. Кроме того, он вычисляет, насколько эта дата удалена от сегодняшнего дня. Код понятен без пояснений, мы приведем его здесь для полноты картины.

Листинг 13.13. Утилиты для работы с датами

```
public class Utils
{
   private static String tag = "Utils";
   public static Date getDate(String dateString)
   throws ParseException {
      DateFormat a = getDateFormat();
      Date date = a.parse(dateString);
      return date;
   }
   public static String test(String sdate){
      try {
         Date d = getDate(sdate);
         DateFormat a = getDateFormat();
         String s = a.format(d);
         return s;
      }
      catch(Exception x){
         return "problem with date:" + sdate;
      }
   }
   public static DateFormat getDateFormat(){
      SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
      // ФорматДаты df = DateFormat.getDateInstance(DateFormat.SHORT);
      df.setLenient(false);
      return df;
   }

   // валидные даты: 1/1/2009, 11/11/2009,
   // невалидные даты: 13/1/2009, 1/32/2009
   public static boolean validateDate(String dateString){
      try {
         SimpleDateFormat df = new SimpleDateFormat("MM/dd/yyyy");
         df.setLenient(false);
         Date date = df.parse(dateString);
         return true;
      }
      catch(ParseException x) {
         return false;
      }
   }
   public static long howfarInDays(Date date){
      Calendar cal = Calendar.getInstance();
      Date today = cal.getTime();
      long today_ms = today.getTime();
      long target_ms = date.getTime();
      return (target_ms - today_ms)/(1000 * 60 * 60 * 24);
   }
}
```

Теперь рассмотрим реализацию конфигурирующего явления, о котором мы уже упоминали.

Реализация явления для конфигурации виджетов

В разделе «Архитектура виджетов основного экрана» мы объяснили роль конфигурирующего явления и его функции. В примере с виджетом, напоминающим о днях рождения, эти функции реализованы в классе явления, который называется ConfigureBDayWidgetActivity. Исходный код этого класса приведен в листинге 13.14.

Этот класс собирает следующие данные: имя человека и следующий день рождения. Затем он создает BDayWidgetModel и сохраняет ее в совместно используемых настройках. Кроме того, в нем есть функция, приспособленная для передачи BDayWidgetModel соответствующему виду виджета.

Листинг 13.14. Реализация конфигурирующего явления

```
public class ConfigureBDayWidgetActivity extends Activity
{
   private static String tag = "ConfigureBDayWidgetActivity";
   private int mAppWidgetId = AppWidgetManager.INVALID_APPWIDGET_ID;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.edit_bday_widget);
      setupButton();

      Intent intent = getIntent();
      Bundle extras = intent.getExtras();
      if (extras != null) {
         mAppWidgetId = extras.getInt(
            AppWidgetManager.EXTRA_APPWIDGET_ID,
            AppWidgetManager.INVALID_APPWIDGET_ID);
      }

   }

   private void setupButton(){
      Button b =
         (Button)this.findViewById(R.id.bdw_button_update_bday_widget);
      b.setOnClickListener(
         new Button.OnClickListener(){
            public void onClick(View v)
            {
               parentButtonClicked(v);
            }
         });

   }
   private void parentButtonClicked(View v){
      String name = this.getName();
      String date = this.getDate();
      if (Utils.validateDate(date) == false){
```

```
            this.setDate("wrong date:" + date);
            return;
        }
        if (this.mAppWidgetId == AppWidgetManager.INVALID_APPWIDGET_ID){
            return;
        }
        updateAppWidgetLocal(name,date);
        Intent resultValue = new Intent();
        resultValue.putExtra(
                AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
        setResult(RESULT_OK, resultValue);
        finish();
    }
    private String getName(){
        EditText nameEdit =
                (EditText)this.findViewById(R.id.bdw_bday_name_id);
        String name = nameEdit.getText().toString();
        return name;
    }
    private String getDate(){
        EditText dateEdit =
                (EditText)this.findViewById(R.id.bdw_bday_date_id);
        String dateString = dateEdit.getText().toString();
        return dateString;
    }
    private void setDate(String errorDate){
        EditText dateEdit =
                (EditText)this.findViewById(R.id.bdw_bday_date_id);
        dateEdit.setText("error");
        dateEdit.requestFocus();
    }
    private void updateAppWidgetLocal(String name, String dob){
        BDayWidgetModel m = new BDayWidgetModel(mAppWidgetId,name,dob);
        updateAppWidget(this,AppWidgetManager.getInstance(this),m);
        m.savePreferences(this);
    }

    public static void updateAppWidget(Context context,
            AppWidgetManager appWidgetManager,
            BDayWidgetModel widgetModel)
    {
        RemoteViews views = new RemoteViews(context.getPackageName(),
                R.layout.bday_widget);

        views.setTextViewText(R.id.bdw_w_name
            , widgetModel.getName() + ":" + widgetModel.iid);

        views.setTextViewText(R.id.bdw_w_date
            , widgetModel.getBday());

        // обновление имени
```

```
        views.setTextViewText(R.id.bdw_w_days,Long.toString(
                widgetModel.howManyDays()));

            Intent defineIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://www.google.com"));
            PendingIntent pendingIntent =
                PendingIntent.getActivity(context,
                    0 /* без КодаЗапроса*/,
                    defineIntent,
                    0 /* без флагов */);
            views.setOnClickPendingIntent(R.id.bdw_w_button_buy,
                        pendingIntent);

        // указание диспетчера виджетов
        appWidgetManager.updateAppWidget(widgetModel.iid, views);
    }
}
```

Если рассмотреть код функции updateAppWidgetLocal(), становится ясно, что эта функция создает модель и сохраняет ее. Затем используется функция updateAppWidget() для отображения виджета. Необходимо отметить, что с функцией updateAppWidget() применяется отложенное намерение, регистрирующее обратный вызов. Отложенное намерение принимает основное следующим образом:

```
Intent defineIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://www.google.com"));
```

и создает отложенное намерение для запуска явления. Правда, отложенные намерения с тем же успехом можно использовать и для запуска служб. Необходимо также отметить, что эта функция может взаимодействовать с RemoteViews и AppWidgetManager. Особо остановимся на таких чертах этой функции, как:

- получение RemoteViews из шаблона;
- установление текстовых значений для RemoteViews;
- регистрация отложенного намерения посредством RemoteViews;
- активация AppWidgetManager для отправки RemoteViews виджету;
- возвращение результата в конце.

ПРИМЕЧАНИЕ

Статическую функцию updateAppWidget можно вызывать откуда угодно, если известен ID виджета. Таким образом, понятно, что можно обновлять виджет из любого места и процесса, действующего в устройстве, как визуального, так и невизуального.

Для завершения явления важно использовать следующий код:

```
Intent resultValue = new Intent();
resultValue.putExtra(AppWidgetManager.EXTRA_APPWIDGET_ID, mAppWidgetId);
setResult(RESULT_OK, resultValue);
finish();
```

Обратите внимание: мы передаем ID виджета обратно вызывающей стороне. Благодаря этому AppWidgetManager узнает, что конфигурирующее явление завершило работу с этим экземпляром виджета.

В заключение нашего разговора о конфигурации виджетов представим шаблон формы для конфигурирующего явления виджета; этот шаблон приведен в листинге 13.15. В этом виде нет ничего сложного: несколько текстовых полей и элементов для редактирования, а также кнопка для обновлений. Шаблон был показан на рис. 13.4.

Листинг 13.15. Определение шаблона для конфигурирующего явления

```
<!-- res/layout/edit_bday_widget.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/root_layout_id"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/bdw_text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Name:"
   />
<EditText
   android:id="@+id/bdw_bday_name_id"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Anonymous"
   />
<TextView
   android:id="@+id/bdw_text2"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Birthday (9/1/2001):"
   />
<EditText
   android:id="@+id/bdw_bday_date_id"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="ex: 10/1/2009"
   />
<Button
   android:id="@+id/bdw_button_update_bday_widget"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="update"
/>
</LinearLayout>
```

На этом мы завершаем разговор о внедрении виджетов. В данном упражнении были рассмотрены следующие моменты:

- определение виджета;
- отклик на обратные вызовы виджета;
- предоставление конфигурирующего явления для виджета;
- демонстрация использования `RemoteViews`;
- предоставление фреймворка для управления состояниями;
- разработка красивого шаблона для виджета.

Теперь мы дадим некоторые рекомендации, касающиеся работы с виджетами.

Ограничения и дополнения, связанные с виджетами

На первый взгляд виджеты домашней страницы Android кажутся несложными. Однако с ними связано множество нюансов, на которые следует обращать внимание при их написании — это дело пока еще является недостаточно освоенным.

Если ваш виджет не требует никакого управления состояниями и инициировать его нужно будет не чаще нескольких раз в день, то написать такой виджет очень просто.

При создании виджетов более высокого уровня потребуется управлять состояниями (как в нашем примере), но такой уровень используется нечасто. При работе с подобными виджетами будет полезен фреймворк управления состояниями. В этой главе был показан лишь каркас данного фреймворка. Предполагаем, что вы сможете найти или самостоятельно написать варианты, которые будут не только более многофункциональны, но и более гибки и надежны.

Еще более совершенные виджеты должны активироваться в течение секунд и даже миллисекунд. При работе с такими виджетами вам придется применить собственные вызовы обновлений (update calls) при помощи менеджера оповещений (alarm manager). Кроме того, вам понадобится служба для управления частыми изменениями состояний, одного только фреймворка долговременных настроек будет недостаточно. Например, если вы собираетесь написать виджет для `StopWatch`, то вам понадобится таймер, считающий как минимум секунды, а также будет нужно отслеживать счетчики, то есть их состояния.

Требуется внимательно отнестись и к тому, что `RemoteViews`, на базе которых работает фреймворк видов виджетов, не приспособлен для непосредственного внесения изменений в виджеты (по крайней мере, в документации об этом ничего не сказано). В `RemoteViews` также присутствуют ограничения, связанные с тем, какие разновидности видов и шаблонов можно использовать. Напрямую управлять видами вы не сможете — только при помощи методов из класса `RemoteViews`.

Учитывая архитектуру и назначение современных виджетов, Google предполагает, что большинство виджетов будет относиться к первому и второму уровням сложности. Существует масса возможностей расширения фреймворка виджетов в будущих версиях.

Ресурсы

В процессе подготовки материала для этой главы мы нашли следующие ресурсы, которые считаем полезными. Они перечислены в порядке убывания важности.

- Официальная документация SDK Android по программированию виджетов: http://developer.android.com/guide/topics/appwidgets/index.html.
- Для управления состояниями необходимо понимать API SharedPreferences. Ссылка на API для этого класса: http://developer.android.com/reference/android/content/SharedPreferences.html.
- При работе с совместно используемыми настройками важен API SharedPreferences.Editor. Он доступен по ссылке http://developer.android.com/reference/android/content/SharedPreferences.Editor.html.
- По следующей ссылке с сайта Android рассказано, как создавать красивые шаблоны виджетов: http://developer.android.com/guide/practices/ui_guidelines/widget_design.html.
- Для рисования видов виджетов и управления ими необходимо понимать API RemoteViews. Этот API находится здесь: http://developer.android.com/reference/android/widget/RemoteViews.html.
- Сами виджеты управляются классом диспетчера виджетов. API этого класса находится по адресу: http://developer.android.com/reference/android/appwidget/AppWidgetManager.html.
- Если вам срочно нужно позаимствовать немного кода, чтобы начать работать с виджетами, можете посетить сайт одного из авторов книги, где собраны полезные фрагменты кода: http://www.satyakomatineni.com/akc/display?url=DisplayNoteIMPURL&reportId=3300&ownerUserId=satya.
- Кроме того, по следующей ссылке мы разместили исследовательские заметки, которые писали параллельно с этой главой: http://www.satyakomatineni.com/akc/display?url=DisplayNoteIMPURL&reportId=3299&ownerUserId=satya.

Резюме

Нам было очень интересно писать эту главу и исследовать возможности, связанные с виджетами основного экрана Android. При всей простоте виджеты могут в очень значительной мере помочь пользователю.

Мы изучили теоретические основы виджетов и показали рабочий пример, на котором стали понятнее различные нюансы. Мы рассказали, зачем нужны модели виджетов и управление состояниями виджетов. Надеемся, что показанный нами код, предназначенный для управления состояниями, пригодится вам при разработке собственных виджетов. Наконец, мы коснулись проблем разработки и ограничений, связанных с виджетами.

14 Поиск в Android

В двух последних главах мы рассказали о двух инновациях Android, тесно связанных с основным экраном устройства. В главе 12 мы изучили, как располагать на главной странице живые каталоги и обеспечивать быстрый доступ к изменяющимся данным поставщиков содержимого. В главе 13 были рассмотрены виджеты основного экрана, выводящие мгновенные снимки важной информации прямо на основной экран.

Продолжая тему «информации на кончиках пальцев» (information at your fingertips), в этой главе мы изучим поисковый фреймворк, действующий в Android. Поисковый фреймворк Android очень обширен. Хотя на первый взгляд кажется, что поиск Android действует только на основном экране устройства, поисковые функции могут применяться и к явлениям вашей программы.

Начнем с общего рассмотрения поисковой функции Android. Мы расскажем о глобальном поиске (global search), живом поиске (search suggestion), замене вариантов (suggestion rewriting) и поиске в веб. Вы узнаете, как включать программы в глобальный поиск и исключать их из него. В ходе данного исследования юзабилити (удобства использования) мы покажем, как поставщики поиска (suggestion providers) взаимодействуют с глобальным поиском.

После изучения юзабилити мы рассмотрим, как интегрировать явления программы с поисковым ключом (search key). Поработаем с явлениями, которые не предназначались именно для поиска, а также продемонстрируем явление, деактивирующее поиск. Кроме того, мы рассмотрим феномен «введите, что нужно найти» (type-to-search) — функцию, которая может использоваться явлениями в программах для активации поиска. Мы также покажем, как напрямую активировать поиск через меню.

Расширяемость поиска в Android достигается благодаря использованию функции, называемой *поставщиком поиска* (suggestion provider). Мы исследуем этот феномен и напишем простой поставщик поиска, наследующий свойства базового поставщика, который входит в состав Android.

Зачастую вам придется писать пользовательские поставщики поиска с нуля. Мы обсудим и этот вопрос, проникнув при его рассмотрении в самое сердце поисковой архитектуры Android.

Наконец, мы рассмотрим две сложные темы и покажем, как использовать имеющиеся на устройстве клавиши действия (action keys) для активации пользовательских действий при помощи элементов живого поиска. Вы также узнаете, как передавать данные, специфичные для приложения, при активации поиска. В заключение главы будет приведен список справочных ресурсов.

Опыт поиска в Android

Возможности поиска в Android дополняют функции всем знакомой панели поиска с сайта Google для поиска как информации, расположенной в устройстве, так и контента из веб. Этот механизм поиска также можно использовать для активации программ прямо из поисковой строки, расположенной на основном экране. В Android данный механизм реализуется при помощи поискового фреймворка, с которым могут взаимодействовать и локальные приложения.

Поисковый фреймворк в Android очень прост. В ней используется обычная поисковая строка, в которую пользователь может вводить данные. Он действует и при применении глобального поиска на домашней странице, и при поиске по отдельной программе.

Когда пользователь вводит текст, Android принимает эти данные и передает их различным приложениям, зарегистрированным для отклика на поиск. Приложения отвечают наборами вариантов. Android собирает отклики сразу из нескольких программ и выводит их в виде списка *вариантов* (suggestions).

При нажатии одного из данных вариантов Android активирует программу, предложившую этот вариант. При этом Android использует интегрированный поиск (его также называют обобщенным или федеративным) по группе участвующих в поиске приложений.

Хотя общая идея и проста, детали протокола очень обширны. Эти детали будут изучены на конкретных примерах ниже в данной главе. В этом разделе мы рассмотрим поисковую форму с точки зрения пользователя.

Исследование глобального поиска в Android

Поисковую строку в Android нельзя не заметить: она расположена на основном экране. Поле для поиска находится в левом верхнем углу экрана, вместе с логотипом Google и изображением увеличительного стекла (рис. 14.1). Поисковую строку также называют QSB (Quick Search Box, поле быстрого поиска).

Чтобы начать поиск, можно ввести запрос прямо в это поле. Кроме того, для этого можно нажать клавишу действия Search (Поиск). Клавишами действия (action keys) называются кнопки, показанные в правой части рис. 14.1. Клавиша для поиска обозначена символом лупы. В этой главе мы будем называть ее *поисковой клавишей*. Символ лупы в поисковой строке будет называться *поисковой пиктограммой*.

Поисковую клавишу, как и клавишу Home (Домой), можно нажимать в любой момент, независимо от того, какая программа сейчас отображается на экране. Однако если программа находится в фокусе, поиск в ней можно специализировать — об этом мы поговорим позже. Такой настраиваемый поиск еще называется *локальным*. Более общий, обычный поиск без дополнительных настроек именуется *глобальным*.

Рассмотрим, как работать с полем быстрого поиска. Чтобы это поле попало в фокус, на нем нужно либо специально щелкнуть кнопкой мыши, либо нажать поисковую клавишу. Пока ничего не пишите в этом поле. В данный момент экран Android будет выглядеть примерно так, как показано на рис. 14.2.

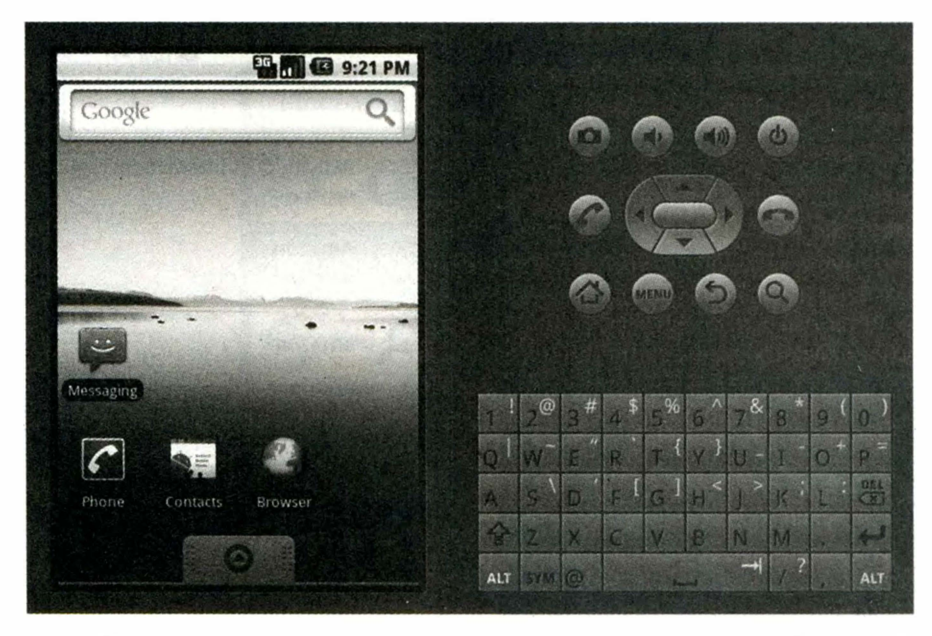


Рис. 14.1. Главная страница Android с полем быстрого поиска и поисковой клавишей

В зависимости от того, как вы использовали устройство ранее, вид изображения с рис. 14.2 может отличаться, поскольку Android угадывает ваши поисковые запросы на основании предыдущих действий в системе. Такой режим работы, при котором в поле поиска не вводится никакой информации, называется *нулевыми вариантами* (zero suggestions). В зависимости от введенного поискового запроса, Android предлагает пользователю несколько вариантов. Эти варианты выстраиваются под поисковой строкой в виде списка. Часто этот механизм называется *живым поиском* (search suggestions). При вводе каждой следующей буквы Android будет изменять список предлагаемых вариантов. Если в поле не введен никакой текст, Android отобразит нулевые варианты. На рис. 14.2 Android определила, что ранее пользователь работал с программой Spare Parts и что такой вариант можно предложить, хотя никакого текста в поисковой строке пока нет. Мы пока ничего не печатали в поле быстрого поиска, но Android, ожидая ввода, уже отображает виртуальную клавиатуру. Она также показана на рис. 14.2.

Посмотрим, что происходит, когда мы начинаем печатать (рис. 14.3). Когда мы вводим в поле быстрого поиска символ a, Android предлагает варианты, начинающиеся с «а» либо как-то связанные с «а». Как видите, Android уже нашла установленные в устройстве программы, названия которых начинаются с «а», а также предложила поиск по веб.

Теперь при помощи кнопки со стрелкой, указывающей вниз, мы выделяем первый вариант. На рис. 14.3 показано, как это делается.

Обратите внимание: когда подсвечен первый вариант, фокус также перемещается с поискового поля на этот вариант. Кроме того, Android активирует полноэкранный режим, скрывая виртуальную клавиатуру, так как при навигации вы не будете набирать. На расширенном экране вы видите больше вариантов.



Рис. 14.2. Поле для быстрого поиска — режим нулевых вариантов

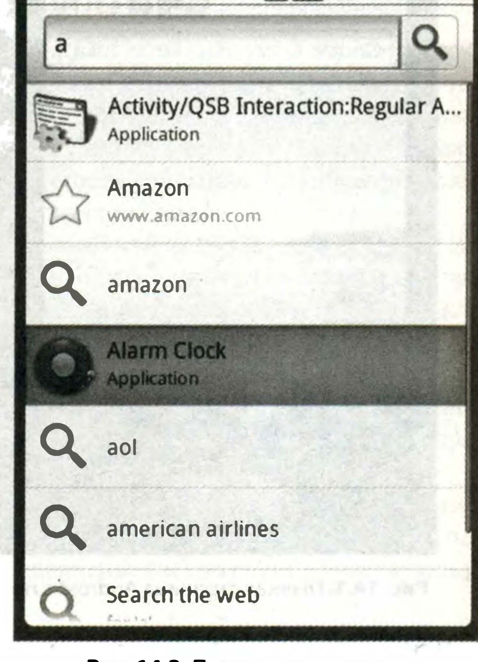


Рис. 14.3. Поисковые варианты

Вновь рассмотрим эти варианты. Android принимает текст, уже введенный в поисковую строку, и ищет так называемые *поставщики поиска* (suggestion providers). Android вызывает каждый из поставщиков поиска в асинхронном режиме, для получения набора соответствующих запросу вариантов в виде набора строк. Android ожидает, что эти строки, называемые поисковыми вариантами, будут соответствовать заданному набору колонок (suggestion columns). Обрабатывая эти уже известные системе колонки, Android строит список вариантов. При изменении поискового запроса Android повторяет эту операцию снова и снова.

ПРИМЕЧАНИЕ

Набор поисковых вариантов также называется курсором вариантов (suggestion cursor). Это объясняется тем, что поставщик содержимого, представляющий поставщик поиска, возвращает объект cursor.

Здесь мы вновь переходим к работе с полем быстрого поиска. Android опять отображает виртуальную клавиатуру. Кроме того, на рис. 14.3 необходимо отметить связь между подсвеченным вариантом и текстом, введенным в поле быстрого поиска. Поисковый текст остался прежним — a, хотя мы уже выбрали конкретный вариант — программу Alarm Clock. Однако на рис. 14.4 показано, что так бывает не всегда. Здесь мы выбрали вариант, указывающий на Amazon.

Обратите внимание, как поисковый запрос a заменяется целой URL, по которой расположен Amazon. Теперь можно либо нажать поисковую пиктограмму в строке

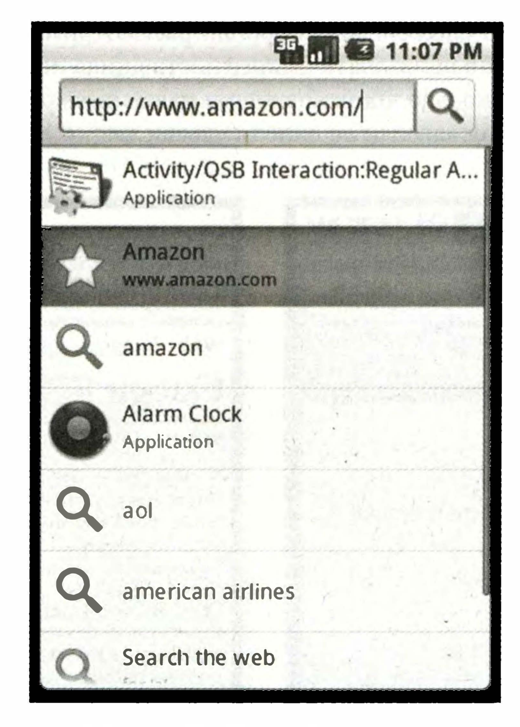


Рис. 14.4. Переписывание вариантов

для быстрого поиска — в результате мы окажемся в Amazon, — либо просто щелкнуть на выделенном запросе. Результат будет одинаковым.

ПРИМЕЧАНИЕ

Процесс изменения поискового запроса на основе выделенного варианта называется переписыванием вариантов (suggestion rewriting).

Переписывание вариантов будет подробно рассмотрено чуть ниже. Если коротко — Android использует одну из колонок курсора вариантов для поиска запрашиваемого текста. Если такая колонка существует, содержащаяся в ней информация будет подставлена на место поискового запроса, если нет — введенный поисковый запрос останется в исходном виде.

Если поисковый вариант не переписывается, существует два варианта развития событий. При нажатии поискового символа на панели Google поисковик будет искать текст, содержащийся в поисковой строке, независимо от предлагаемых вариантов. Если нажать один из предложенных вариантов, то в программе запустится так называемое *поисковое явление*, в которое заносится предложенный запрос, а после этого начинается поиск. Затем это явление отвечает за вывод результатов поиска.

На рис. 14.5 показан пример непосредственной активации одного из вариантов. В этом примере осуществляется поиск программы, которая называется API Demos. При нажатии этого варианта Android сразу запускает эту программу. Сам по себе

этот процесс довольно сложен, мы подробно рассмотрим его ниже в этой главе (см. раздел «Реализация пользовательского поставщика поиска»).

На рис. 14.6 показано, что происходит при нажатии изображения значка лупы на панели быстрого поиска, если на панели указан запрос a.

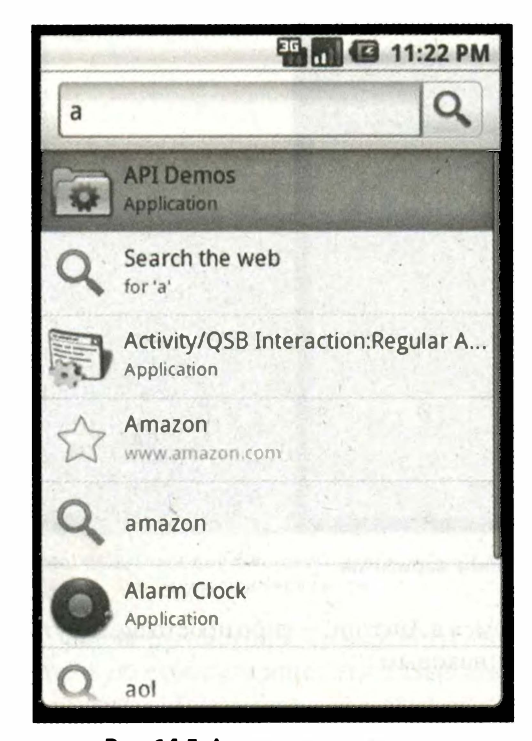


Рис. 14.5. Активация программы из поисковой строки

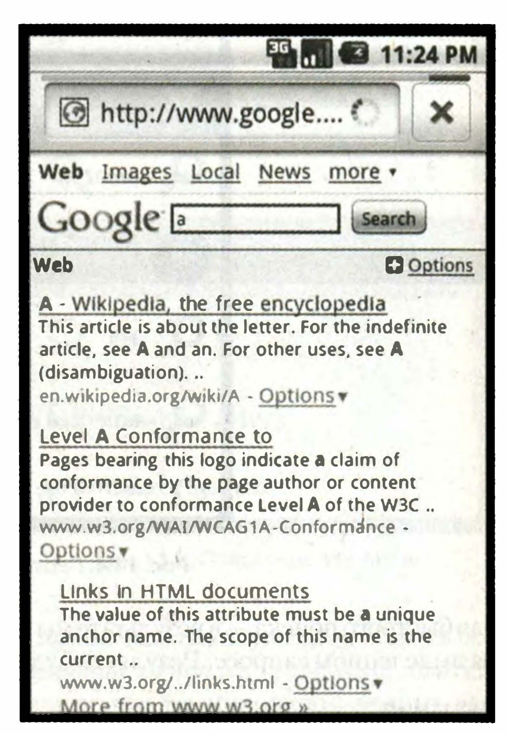


Рис. 14.6. Поиск по Сети

Теперь мы умеем пользоваться строкой для быстрого поиска и нам предстоит узнать, как включать определенные программы в глобальный поиск либо исключать их из него.

Включение поставщиков поиска для глобального использования

Как мы уже говорили, программы используют поставщики поиска (поставщики вариантов) для отклика на поисковые запросы. То, что в нашей программе есть инфраструктура, необходимая для отклика на поисковые запросы, еще не означает, что варианты будут автоматически отображаться на панели быстрого поиска. Пользователь должен допустить ваш поставщик к участию в поиске. На следующих иллюстрациях показаны различные стадии процесса включения или исключения имеющихся поставщиков поиска.

Начнем с экрана настроек Android (рис. 14.7).

Чтобы перейти к этому виду, нужно нажать стрелку Show Applications (Отобразить приложения) в нижней части экрана устройства (см. также рис. 14.1, где изображен основной экран). Стрелка, указывающая вниз, позволяет перейти к программе Settings (Настройки), как это показано на рис. 14.7. После этого вы попадете на страницу настроек Android, изображенную на рис. 14.8.

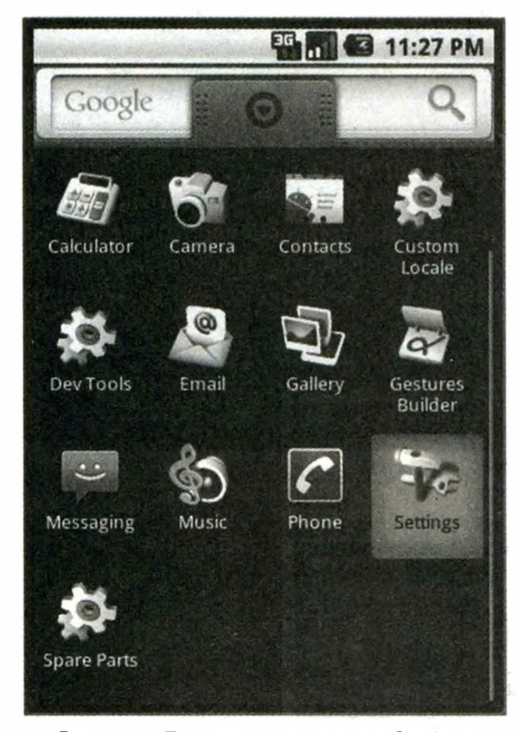


Рис. 14.7. Переход к программе Settings



Рис. 14.8. Переход к настройкам программы Search

Из предложенных настроек Android выберите Search (Поиск). Эта функция управляет настройками и историей поиска. С ее помощью вы попадаете в программу Search settings (Настройки поиска), показанную на рис. 14.9.

В этом явлении особо рассмотрим раздел Quick Search Box (Поле быстрого поиска) и выберем те элементы, по которым может производиться поиск. Таким образом, мы выбираем, что будем искать в телефоне. Отобразится список доступных поставщиков поиска (рис. 14.10).

Simple Search Suggestions Provider — это один из поставщиков вариантов, имеющихся в нашем арсенале. Позже, в одном из примеров мы напишем код к этому поставщику поиска. По умолчанию новый поставщик вариантов не подсвечивается. Щелкните на этом элементе строки, чтобы подключить его к поиску. Когда он будет активирован, вид этой страницы изменится следующим образом — один из элементов будет выбран (рис. 14.11).

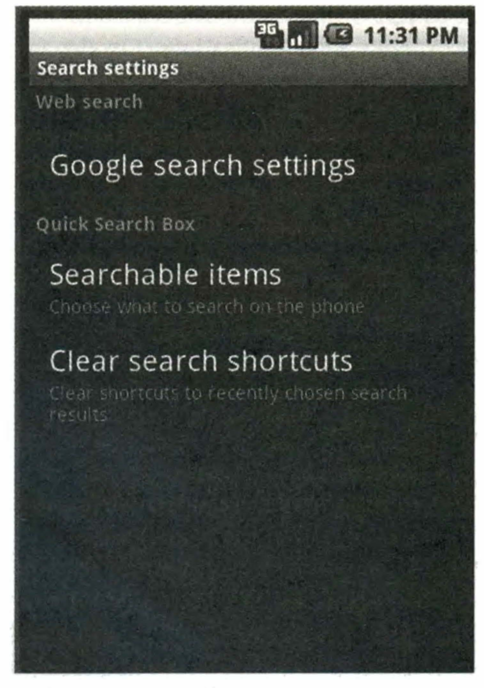


Рис. 14.9. Программа настроек поиска

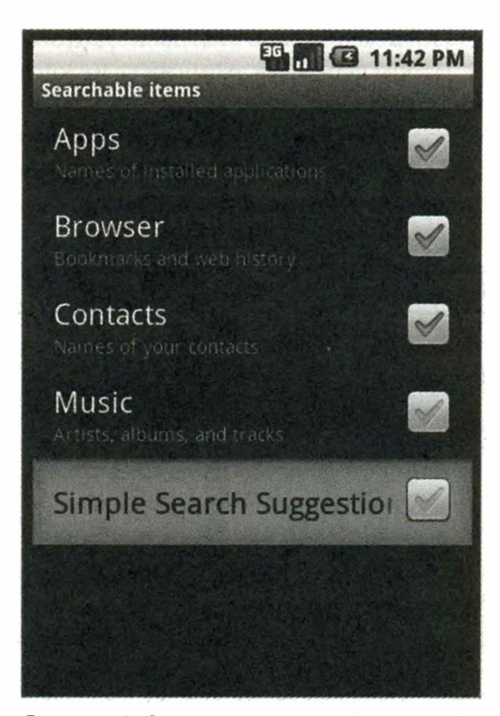


Рис. 14.10. Отключенная программа поиска поставщиков вариантов

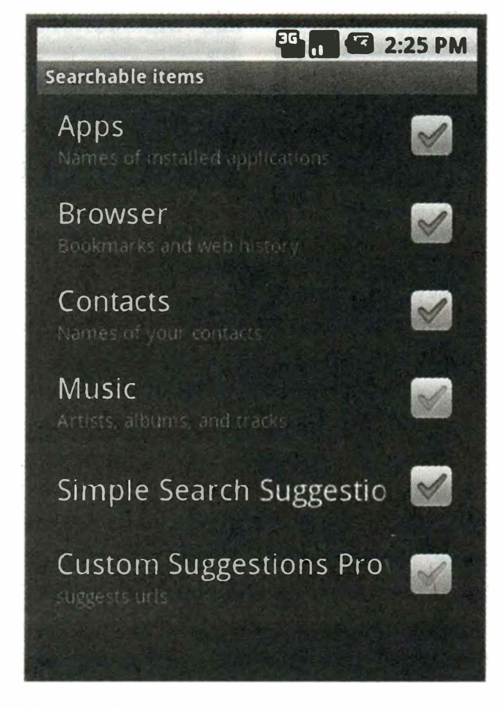


Рис. 14.11. Включенная программа поставщика вариантов поиска

Взаимодействие поставщиков поиска и поля быстрого поиска (QSB)

Теперь мы рассмотрим, как варианты, предлагаемые поставщиком поиска, используются в QSB. Обычно варианты от нового поставщика поиска отображаются как More results (Дополнительные результаты) под соответствующей пиктограммой, расположенной в списке вариантов (рис. 14.12).

Обратите внимание, как Android собирает большое количество вариантов от нескольких приложений в общем элементе для показа вариантов. При его нажатии Android разворачивает этот элемент (рис. 14.13).



Рис. 14.12. Дополнительные результаты от новых поставщиков поиска

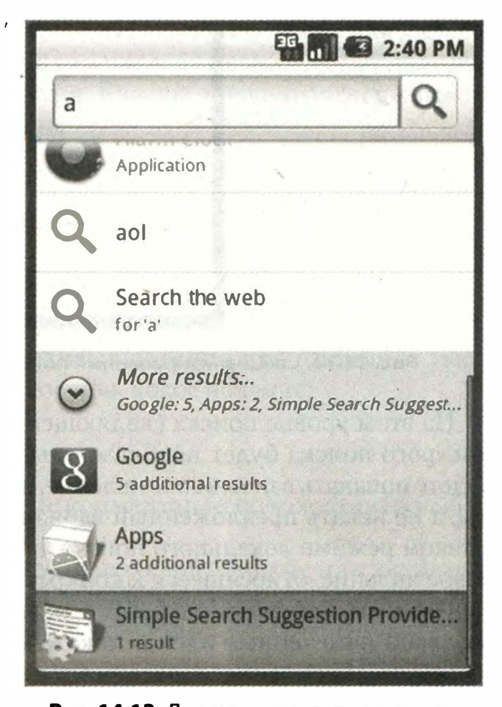


Рис. 14.13. Дополнительные результаты от отдельно взятого поставщика поиска

Обратите внимание — варианты от конкретных программ по-прежнему собираются в группы, но теперь общий список вариантов разбит на группы, соответствующие каждому отдельно взятому приложению. Теперь, если нажать одно из этих приложений, Android инициирует специализированный поиск, в который будут включены варианты, поступившие только от этого приложения. Пример показан на рис. 14.14.

Здесь поиск превращается из глобального в локальный. Такой поиск производится только по программе, предложившей определенный вариант.

ПРИМЕЧАНИЕ

Далее в этой главе, в разделах «Реализация простого поставщика поиска» и «Реализация пользовательского поставщика поиска», мы вернемся к этим скриншотам и рассмотрим их более подробно.

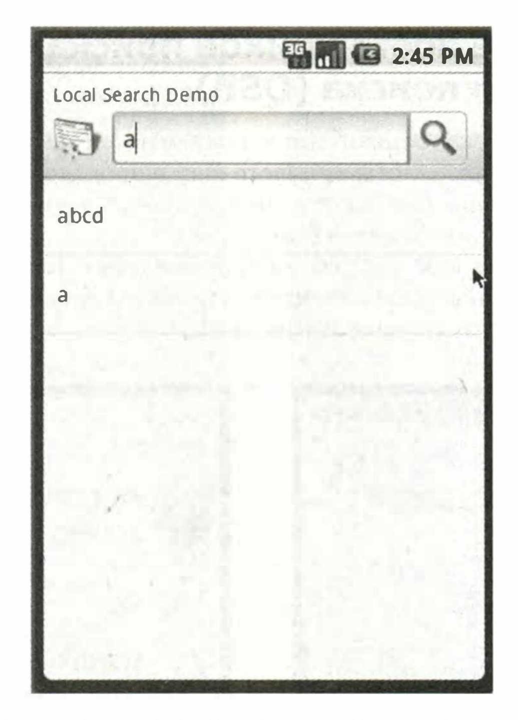


Рис. 14.14. Специализированный поиск по определенному поставщику вариантов

На этом уровне поиска (являющегося локальным) при нажатии пиктограммы быстрого поиска будет использоваться текст поискового запроса, после чего вы будете попадать в поисковое явление, определенное в рамках конкретной программы, а не искать предложенный вариант в веб. А если выбрать один из вариантов в таком режиме локального поиска, вы, опять же, перейдете в специальное поисковое явление, относящееся к конкретной программе.

Пока мы в общем виде рассмотрели принцип работы поиска в Android. Далее разовьем намеченные идеи и покажем вам все практические аспекты поиска на примерах. Начнем с исследования того, как организовано взаимодействие простых явлений и поиска.

Взаимодействие явлений и поисковых клавиш

Что происходит при нажатии пользователем поисковой клавиши в тот момент, когда определенное явление находится в фокусе? Мы рассмотрим работу явлений таких типов, как:

- обычное явление, которому «неизвестно» о том, что идет поиск;
- явление, напрямую отключающее поиск;
- явление, напрямую активирующее глобальный поиск;
- явление, задающее локальный поиск.

Эти варианты будут изучены на рабочих образцах, содержащих следующие основные файлы Java (изучив каждый из них, мы покажем, как конкретные концепции реализуются на экране, в процессе работы программы):

- RegularActivity.java;
- NoSearchActivity.java;
- SearchInvokerActivity.java;
- LocalSearchEnabledActivity.java;
- SearchActivity.java.

Каждый из этих файлов, кроме последнего (SearchActivity.java), является представителем определенного типа явлений, которые будут рассмотрены по вышеописанной схеме. Последний файл, SearchActivity.java, нужен для явления LocalSearchEnabledActivity. Все эти явления, в том числе SearchActivity, используют простые шаблоны с текстовыми видами внутри. Каждое явление поддерживается с применением следующих файлов шаблонов:

- res/layout/main.xml (для RegularActivity);
- res/layout/no_search_activity.xml;
- res/layout/search_invoker_activity.xml;
- res/layout/local_search_enabled_activity.xml;
- res/layout/search_activity.xml.

Два следующих файла определяют указанные явления для Android, а также ищут по метаданным единственное локальное поисковое явление:

- manifest.xml;
- xml/searchable.xml.

В следующем файле содержатся текстовые комментарии по каждому из шаблонов: res/values/strings.xml.

В двух следующих файлах представлены меню, необходимые для активации явлений, а также при необходимости — глобального поиска:

- res/menu/main_menu.xml;
- res/menu/search_invoker_menu.xml.

Далее мы исследуем взаимодействие между явлениями и поисковой клавишей, изучим код этих файлов для каждого типа явления. Для начала рассмотрим работу поисковой клавиши с обычным явлением Android.

Работа поисковой клавиши с обычным явлением

Чтобы узнать, что происходит при нажатии поисковой клавиши, когда в фокусе находится явление, не знающее о поиске, рассмотрим пример с обычным явлением. В листинге 14.1 показан исходный код Java, соответствующий RegularActivity.

Листинг 14.1. Исходный код обычного явления

```
// имя файла: RegularActivity.java
public class RegularActivity extends Activity
{
```

```
private final String tag = "RegularActivity";

@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);
}

@Override
public boolean onCreateOptionsMenu(Menu menu)
{
   // вызов родительского элемента
   // для прикрепления меню с любого уровня системы
   super.onCreateOptionsMenu(menu);
   MenuInflater inflater = getMenuInflater(); //from activity
   inflater.inflate(R.menu.main_menu, menu);
   return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
   appendMenuItemText(item);
   if (item.getItemId() == R.id.menu_clear) {
      this.emptyText();
      return true;
   }

   if (item.getItemId() == R.id.mid_no_search) {
      this.invokeNoSearchActivity();
      return true;
   }
   if (item.getItemId() == R.id.mid_local_search) {
      this.invokeLocalSearchActivity();
      return true;
   }
   if (item.getItemId() == R.id.mid_invoke_search) {
      this.invokeSearchInvokerActivity();
         return true;
   }
   return true;
}

private TextView getTextView()
{
   return (TextView)this.findViewById(R.id.text1);
}

private void appendMenuItemText(MenuItem menuItem)
{
   String title = menuItem.getTitle().toString();
```

```
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + title);
    }
    private void emptyText()
    {
        TextView tv = getTextView();
        tv.setText("");
    }
    private void invokeNoSearchActivity()
    {
        Intent intent = new Intent(this,NoSearchActivity.class);
        startActivity(intent);
    }
    private void invokeSearchInvokerActivity()
    {
        Intent intent = new Intent(this,SearchInvokerActivity.class);
        startActivity(intent);
    }
    private void invokeLocalSearchActivity()
    {
        Intent intent = new Intent(this,LocalSearchEnabledActivity.class);
        startActivity(intent);
    }
}
```

В этом примере показан алгоритм работы простого явления, никак не связанного с поиском. Но в данном примере это явление одновременно является движущим фактором, активирующим явления других типов, которые мы также собираемся протестировать. Вот почему в примере были использованы некоторые элементы меню, представляющие эти дополнительные явления. Каждая функция, начинающаяся с `invoke...`, содержит код, необходимый для запуска интересующих нас явлений других типов.

Чтобы понять, как определяется это явление (листинг 14.2), обратимся к файлу описания. Здесь также показаны определения других явлений, хотя они и будут рассмотрены позже.

Листинг 14.2. Взаимодействие явления и клавиши поиска: файл описания

```
// имя файла: manifest.xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ai.android.search.nosearch">
<application android:icon="@drawable/icon"
    android:label="Test Activity QSB Interaction">
    <activity android:name=".RegularActivity"
        android:label="Activity/QSB Interaction:Regular Activity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <activity android:name=".NoSearchActivity"
```

```
      android:label="Activity/QSB Interaction::Disabled Search">
   </activity>

   activity android:name=".SearchInvokerActivity"
      android:label="Activity/QSB Interaction::Search Invoker">
   </activity>

   <activity android:name=".LocalSearchEnabledActivity"
      android:label="Activity/QSB Interaction::Local Search">
      <meta-data android:name="android.app.default_searchable"
         android:value=".SearchActivity" />
   </activity>

   <activity android:name=".SearchActivity"
      android:label="Activity/QSB Interaction::Search Results">
      <intent-filter>
         <action android:name="android.intent.action.SEARCH" />
         <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
               android:resource="@xml/searchable" />
   </activity>
<!--
   <meta-data android:name="android.app.default_searchable"
            android:value="*" />
-->
</application>
<uses-sdk android:minSdkVersion="4" />
</manifest>
```

Обратите внимание: RegularActivity определяется как основное явление данного примера и не имеет никаких характеристик, связанных с поиском.

Шаблон для этого явления показан в листинге 14.3.

Листинг 14.3. Файл шаблона для обычного явления

```
// имя файла: layout/main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/regular_activity_prompt"
   />
</LinearLayout>
```

Это явление будет выглядеть на экране так, как показано на рис. 14.15.

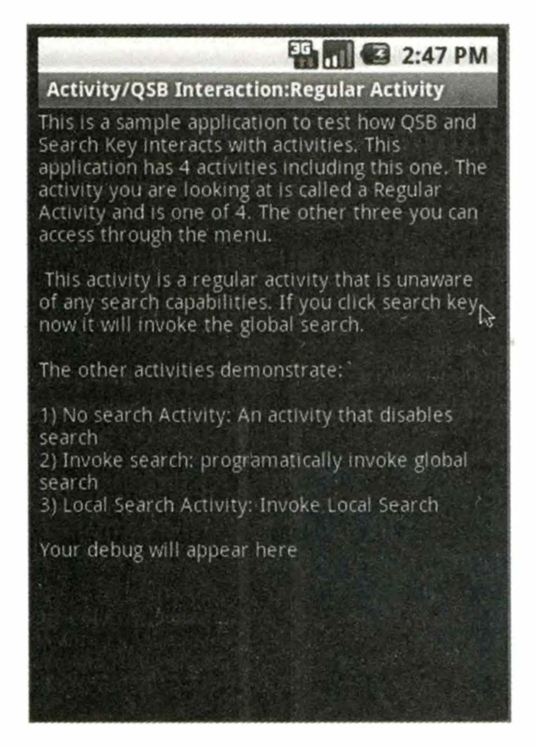


Рис. 14.15. Взаимодействие явления и клавиши поиска

В листинге 14.4 показан файл `strings.xml`, обеспечивающий отображение текста, который вы видите в этом явлении.

Листинг 14.4. Взаимодействие явления и клавиши поиска: файл strings.xml

```
// имя файла /res/values/strings.xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
<!-
****************************************************
* regular_activity_prompt
****************************************************
-->
   <string name="regular_activity_prompt">
   При помощи этой программы можно протестировать,
   как окно быстрого поиска и клавиша поиска
   взаимодействуют с явлениями. В этой программе всего 4 явления,
   включая это. Нас интересует явление, которое
   называется Regular Activity и является одним из 4.
   Доступ к трем другим явлениям
   осуществляется через меню.
```

```
    \n\n
    Это обычное явление, не знающее ни о каких
    поисковых функциях. Если сейчас нажать клавишу поиска,
    то начнется глобальный поиск по Сети.
    \n
    \nВ других явлениях демонстрируется:`
    \n\n1) No search Activity: явление, деактивирующее поиск
    \n2) Invoke search:
    \ инициирование глобального поиска программными средствами
    \n3) Local Search Activity: Инициирование локального поиска
    \n
    \nЗдесь будет осуществляться отладка
    </string>

<!--
**************************************************
* no_search_activity_prompt
**************************************************
-->
    <string name="no_search_activity_prompt">
    В этом явлении onSearchRequested
    возвращает false. Сейчас клавиша поиска
    должна игнорироваться
    \n
    \nМожно нажать "Назад", чтобы попасть
    в предыдущее явление, и вновь при помощи меню
    выбрать другие явления.
    </string>
<!--
**************************************************
* search_activity_prompt
**************************************************
-->
   <string name="search_activity_prompt">
Так вызывается поисковое явление или выводятся результаты поиска.
Это явление инициируется при нажатии поисковой клавиши,
когда другое явление использует это явление
для вывода собственных результатов поиска
\n\n
Обычно можно получить строку запроса от
намерения, чтобы просмотреть, каков был запрос.
   </string>
<!--
**************************************************
* search_invoker_activity_prompt
**************************************************
-->
<string name="search_invoker_activity_prompt">
В этом явлении элемент поискового меню используется
```

```
для активации поиска, заданного по умолчанию. В данном случае, поскольку
в этом явлении не предусмотрен локальный поиск,
инициируется глобальный поиск. При помощи кнопки
меню можно открыть поисковое меню. При нажатии
этого меню начнется
глобальный поиск.
</string>
<!--
**************************************************
* local_search_enabled_activity_prompt
**************************************************
-->
   <string name="local_search_enabled_activity_prompt">
Это очень простое явление, указывающее в файле описания на то,
что с ним связано поисковое явление.
Если при наличии такой связи нажимается
клавиша поиска, то вместо глобального поиска начинается локальный.
\n\n
Локальная природа этого явления понятна по метке,
расположенной в поле быстрого поиска, и по находящейся там же подсказке.
Вся эта информация получена из метаданных поиска.
\n\n
При щелчке на пиктограмме запроса вы переходите
к локальному поисковому явлению.
   </string>
<!--
**************************************************
* Другие значения
**************************************************
   <string name="app_name">Sample Search Application</string>
-->
   <string name="search_label">Local Search Demo</string>
   <string name="search_hint">Local Search Demo Hint</string>
</resources>
```

Как и файл описания Android, файл strings.xml обслуживает все явления данного проекта. Как видите, строковая константа regular_activity, находящаяся в strings.xml, указывает на текст, который вы видите в регулярном явлении.

В листинге 14.5 показан XML-файл меню, используемый с обычным явлением. Это меню в действии показано ниже, на рис. 14.16.

Листинг 14.5. Файл меню обычного явления

```
// имя файла: /res/menu/main_menu.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <!-- В этой группе используется категория, заданная по умолчанию. -->
   <group android:id="@+id/menuGroup_Main">
      <item android:id="@+id/mid_no_search"
         android:title="No Search Activity" />

      <item android:id="@+id/mid_local_search"
```

```
            android:title="Local Search Activity" />

        <item android:id="@+id/mid_invoke_search"
            android:title="Search Invoker Activity" />

        <item android:id="@+id/menu_clear"
            android:title="clear" />
    </group>
</menu>
```

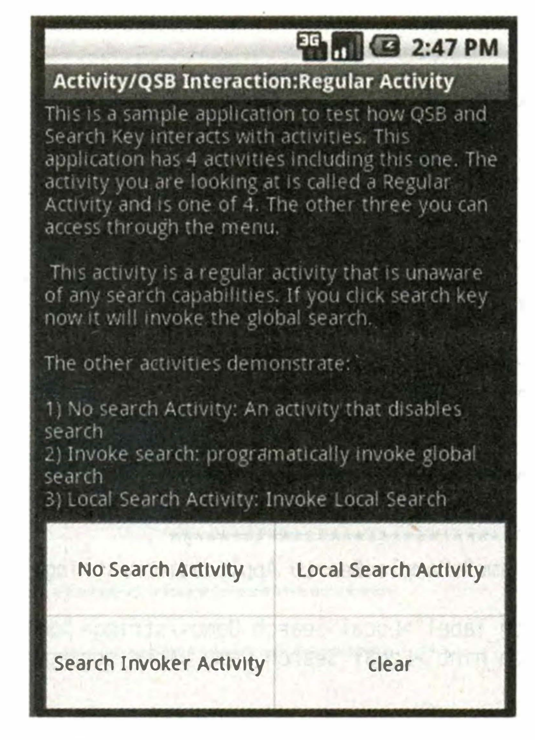


Рис. 14.16. Доступ к другим тестовым явлениям

С помощью этих файлов вы сможете скомпилировать и протестировать данное явление (либо можете подождать, пока будут рассмотрены все явления из данного проекта). Если вы хотите приступить к компиляции прямо сейчас, то остальные явления из файла описания нужно исключить, отметив их знаками комментариев (возможно, у вас еще нет исходного кода для этих явлений).

Теперь, когда явление работает (см. рис. 14.15), нажмите клавишу поиска (на рис. 14.1 показано, где она находится). При нажатии поисковой клавиши начнется глобальный поиск. Ситуация будет такой же, как на рис. 14.2.

СОВЕТ

При работе с явлением, не знающем о существовании локального поиска, запускается глобальный поиск.

Работа с явлением, в котором отключена функция поиска

Как было описано в предыдущем разделе, если работа явления никак не связана с поисковой функцией, то при нажатии клавиши **Search** (Поиск) запускается глобальный поиск. Однако явление может отключить функцию поиска, возвратив в методе обратного вызова `onSearchRequested()` класса явления значение `false`. В листинге 14.6 показан исходный код такого явления.

Листинг 14.6. Явление, отключающее поиск

```
// имя файла: NoSearchActivity.java
public class NoSearchActivity extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.no_search_activity);
      return;
   }
   @Override
   public boolean onSearchRequested()
   {
   return false;
   }
}
```

В листинге 14.7 показан файл шаблона этого явления.

Листинг 14.7. XML-файл явления NoSearchActivity

```
// имя файла: layout/no_search_activity.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/no_search_activity_prompt"
   />
</LinearLayout>
```

Явление `NoSearchActivity` активируется нажатием элемента меню **No Search Activity** (Без поискового явления), показанного на рис. 14.16. Когда это явление отображается, оно выглядит как на рис. 14.17. Если теперь нажать поисковую клавишу (см. рис. 14.1), она не подействует и ничего не произойдет, как будто вы ее и не нажимали.

СОВЕТ

Если мы имеем дело с явлением, в котором отключается поиск, то при нажатии поисковой клавиши деактивируется запуск любого поиска, в том числе глобального.

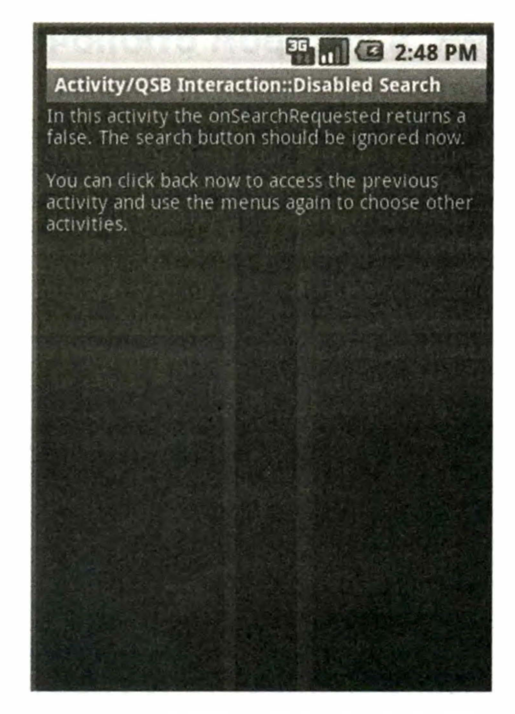


Рис. 14.17. Явление с отключенным поиском

Инициирование поиска через меню

Явление может не только реагировать на нажатие поисковой клавиши, но и использовать возможность непосредственного инициирования поиска через элемент меню поиска. В листинге 14.8 показан исходный код образца явления, выполняющего такую функцию.

Листинг 14.8. SearchInvokerActivity

```
// имя файла: SearchInvokerActivity.java
public class SearchInvokerActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.search_invoker_activity);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu)
   {
      super.onCreateOptionsMenu(menu);
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.search_invoker_menu, menu);
      return true;
   }
   @Override
```

```
    public boolean onOptionsItemSelected(MenuItem item)
    {
        appendMenuItemText(item);
        if (item.getItemId() == R.id.mid_si_clear)
        {
            this.emptyText();
            return true;
        }
        if (item.getItemId() == R.id.mid_si_search)
        {
            this.invokeSearch();
            return true;
        }
        return true;
    }

    private TextView getTextView()
    {
        return (TextView)this.findViewById(R.id.text1);
    }

    private void appendMenuItemText(MenuItem menuItem)
    {
        String title = menuItem.getTitle().toString();
        TextView tv = getTextView();
        tv.setText(tv.getText() + "\n" + title);
    }
    private void emptyText()
    {
        TextView tv = getTextView();
        tv.setText("");
    }
    private void invokeSearch()
    {
        this.onSearchRequested();
    }
}
```

Важнейшие фрагменты исходного кода выделены полужирным шрифтом. Обратите внимание, как ID меню (R.id.mid_si_search) вызывает функцию invokeSearch, которая, в свою очередь, вызывает метод onSearchRequested(). Метод onSearchRequested() инициирует поиск.

В листинге 14.9 показан шаблон этого явления.

Листинг 14.9. XML-шаблон для SearchInvokerActivity

```
// имя файла: layout/search_invoker_activity.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
```

```
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/search_invoker_activity_prompt"
    />
</LinearLayout>
```

В листинге 14.10 показан соответствующий XML-файл меню для этого явления.

Листинг 14.10. XML-меню явления SearchInvokerActivity

```
// имя файла:menu/search_invoker_menu.xml
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <!--В этой группе используется категория, заданная по умолчанию. -->
    <group android:id="@+id/menuGroup_Main">
        <item android:id="@+id/mid_si_search"
            android:title="Search" />

        <item android:id="@+id/mid_si_clear"
            android:title="clear" />
    </group>
</menu>
```

Теперь, когда у нас есть шаблон и меню, рассмотрим рис. 14.18. На нем показано, как выглядит это явление, если вызвать его из основного меню RegularActivity (на рис. 14.16 изображен инициирующий элемент меню).

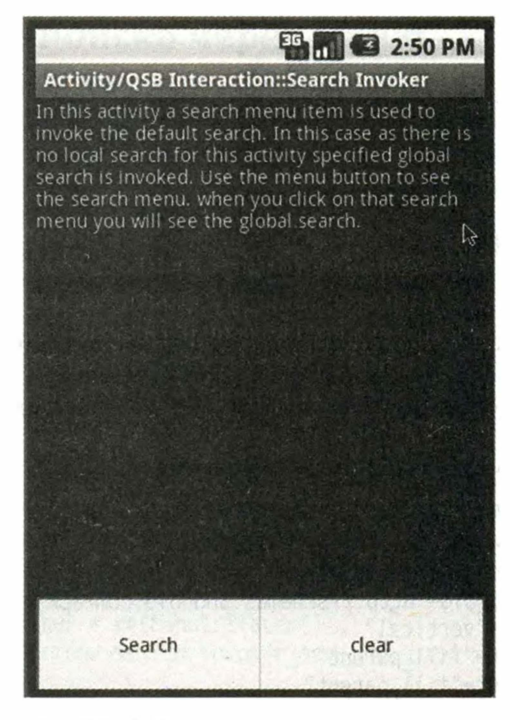


Рис. 14.18. Явление инициирования поиска

Если в этом явлении нажать кнопку Search (Поиск), расположенную в меню, запустится глобальный поиск, как на рис. 14.2.

Локальный поиск и связанные с ним явления

Теперь рассмотрим обстоятельства, в которых поисковая клавиша запускает не глобальный, а локальный поиск. Для этого нам потребуется немного подробнее разобраться с локальным поиском.

Локальный поиск реализуется при помощи трех компонентов. Первый компонент — это поисковое поле, очень похожее на QSB. Поле QSB, предназначенное для локального или для глобального поиска, — это элемент управления для ввода текста. Когда текст введен, нужно нажать поисковую пиктограмму. Обычно поле для локального поиска активируется вместо поля глобального поиска, если в файле описания данного явления объявлено, что оно предназначено для локального поиска. Чтобы отличить QSB для локального поиска от такого же поля для глобального поиска, посмотрите на заголовок QSB (см. заголовок на рис. 14.14) и подсказку (текст, находящийся в поисковом поле до ввода запроса). Как видите, два этих значения берутся из XML-файла с метаданными поиска.

Второй компонент локального поиска — это явление, которое может получать поисковую строку из поля для быстрого локального поиска и показывать набор результатов или любой вывод, касающийся поиска.

Третий, необязательный компонент, участвующий в локальном поиске, — это явление, которому разрешено инициировать только что описанное явление, выводящее результаты поиска (то есть второй компонент). Такое инициирующее явление часто называют *инициатором поиска* (search invoker). Это явление не обязательно использовать, поскольку глобальный поиск может непосредственно инициировать локальный (второе явление-компонент) при нажатии одного из вариантов, предлагаемых при живом поиске.

Эти компоненты и их взаимодействие в общем контексте показаны на рис. 14.19.

На рис. 14.19 показаны и аннотированы важные взаимодействия (номера в кружках), обозначенные стрелками. Ниже этот рисунок объяснен подробно.

- Явление `SearchActivity` должно быть определено в файле описания как явление, способное принимать поисковые запросы. Кроме того, `SearchActivity` использует обязательный XML-файл, где объявляется, как должно выглядеть поле быстрого поиска (в частности, его заголовок, подсказка и т. д.) и связан ли с этим полем поставщик поиска (подробнее это будет показано в листинге 14.12). На рис. 14.19 эти взаимодействия представлены несколькими линиями «Определение» (definition), располагающимися между `SearchActivity` и XML-файлами (файлом описания и файлом метаданных поиска).
- Когда явление `SearchActivity` определено в файле описания (см. листинг 14.2), `Search InvokingActivity` в этом файле служит индикатором связи с `SearchActivity`.
- Имея определение обоих явлений и поместив в фокус явление `SearchInvokingActivity`, мы можем нажать поисковую клавишу. При этом запустится поле для быстрого локального поиска. Этот этап обозначен на рис. 14.19 кружками с цифрами *1* и *2*. Чтобы понять, что поле QSB предназначено для локального поиска, достаточно посмотреть на текст, находящийся в поле до ввода запроса, и на заголовок

поля. Два этих значения задаются в обязательном XML-определении метаданных поиска. Когда поле QSB активируется поисковой клавишей, вы можете вносить в это поле текст запроса. Локальное поле QSB, как и глобальное, поддерживает живой поиск. На рис. 14.19 этот аспект обозначен номером *3*.

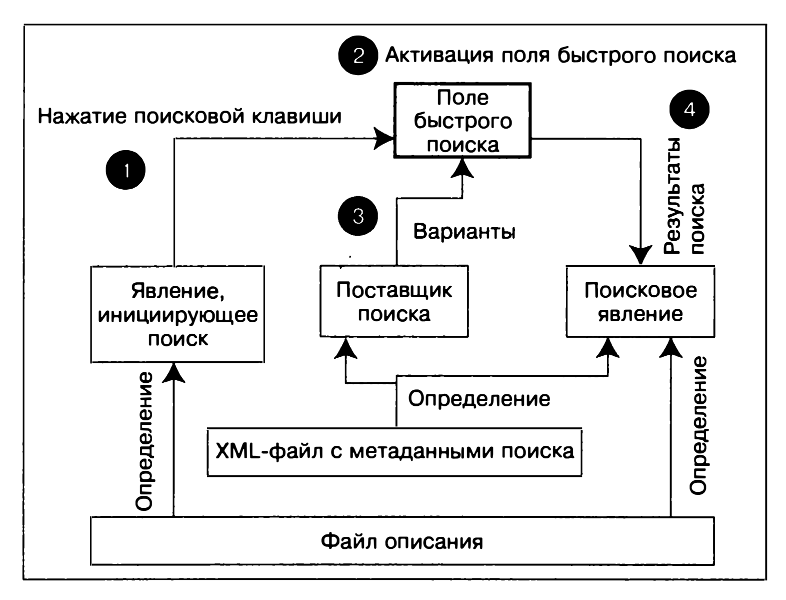


Рис. 14.19. Взаимодействие явлений с локальным поиском

- Когда введен поисковый запрос и нажата пиктограмма поиска, поле передаст поиск явлению `SearchActivity`, которое совершит с ним определенную операцию, например отобразит набор результатов. На рис. 14.19 этот аспект обозначен номером *4*.

Все эти взаимодействия мы изучим на примере исходного кода. Начнем с листинга 14.11, где дан исходный код `SearchActivity` (которое, в свою очередь, отвечает за получение запроса и отображение результатов поиска).

Листинг 14.11. `SearchActivity`

```
// имя файла: SearchActivity.java
public class SearchActivity extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.search_activity);
      return;
   }
}
```

Обратите внимание на то, насколько просто построено поисковое явление. Ниже будет показано, как это явление находит запросы. Пока мы продемонстрируем, как завершает работу активированное явление. Ниже приведен код, на примере кото-

рого видно, как конкретное явление определяется в файле описания в качестве поискового, отвечающего за получение результатов (см. листинг 14.2):

```
<activity android:name=".SearchActivity"
        android:label="Activity/QSB Interaction::Search Results">
      <intent-filter>
      <action android:name="android.intent.action.SEARCH" />
      <category android:name="android.intent.category.DEFAULT" />
   </intent-filter>
   <meta-data android:name="android.app.searchable"
                     android:resource="@xml/searchable" />
</activity>
```

ПРИМЕЧАНИЕ

В поисковом явлении необходимо задать две вещи. Явление должно иметь указатель того, что оно может отвечать на действия SEARCH. Кроме того, нужно указать XML-файл, в котором описываются метаданные, необходимые для взаимодействия с поисковым явлением.

В листинге 14.12 показан XML-файл метаданных для явления SearchActivity.

Листинг 14.12. Searchable.xml: поисковые метаданные

```
///res/xml/searchable.xml
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
   android:label="@string/search_label"
   android:hint="@string/search_hint"
   android:searchMode="showSearchLabelAsBadge"
/>
```

СОВЕТ

Различные параметры, которые могут присутствовать в этом XML-файле, объяснены по адресу http://developer.android.com/reference/android/app/SearchManager.html.

Большинство из этих атрибутов будет рассмотрено ниже в текущей главе. Пока отметим, что атрибут android:label используется в качестве метки (label) поискового поля. Атрибут android:hint применяется для записи в поисковом поле того текста, который отображается до ввода результата (см. рис. 14.14 или 14.21).

Теперь рассмотрим, как любое явление позволяет указывать SearchActivity в качестве цели своего поиска. Для примера возьмем явление, которое использует в качестве цели SearchActivity. Назовем это новое явление LocalSearchEnabledActivity. В листинге 14.13 показан его исходный код.

Листинг 14.13. LocalSearchEnabledActivity

```
// имя файла: LocalSearchEnabledActivity.java
public class LocalSearchEnabledActivity extends Activity
{
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.local_search_enabled_activity);
      return;
   }
}
```

В листинге 14.14 представлен XML-файл шаблона для этого явления.

Листинг 14.14. Файл шаблона LocalSearchEnabledActivity

```
// имя файла: local_search_enabled_activity
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/local_search_enabled_activity_prompt"
   />
</LinearLayout>
```

Это явление можно активировать из основного явления RegularActivity, выбрав в меню команду **Local Search Activity** (Явление локального поиска) (см. рис. 14.16). После того как явление активировано, оно выглядит как на рис. 14.20.

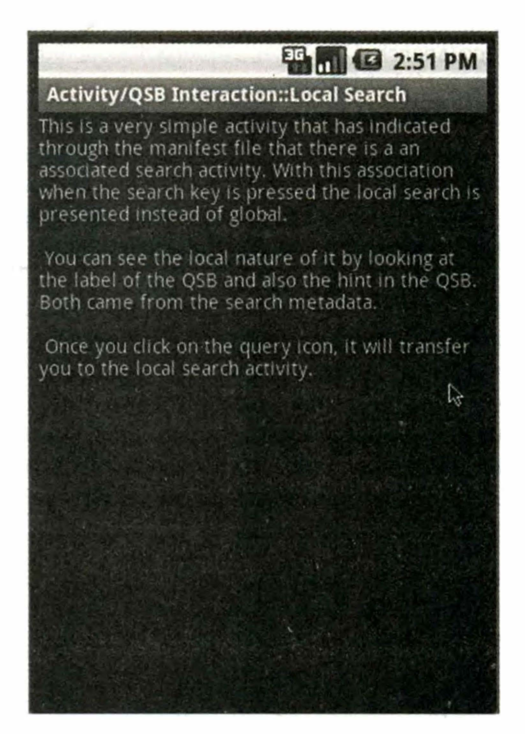


Рис. 14.20. Явление, предназначенное для локального поиска

Если данное явление находится в фокусе, то при нажатии поисковой клавиши (она была показана на рис. 14.1) активируется поле для локального поиска (локальный QSB) (рис. 14.21).

Обратите внимание на метку (label) данного поискового поля и подсказку, находящуюся в этом поле. Посмотрите, как оно отличается от поля глобального поиска (см. рис. 14.2). Метка и подсказка получены из поисковых метаданных, указанных для SearchActivity (см. рис. 14.12). Теперь, если ввести в поисковое поле текст и нажать символ лупы, то в результате активируется SearchActivity (см. листинг 14.11). На рис. 14.22 показано, как выглядит SearchActivity.

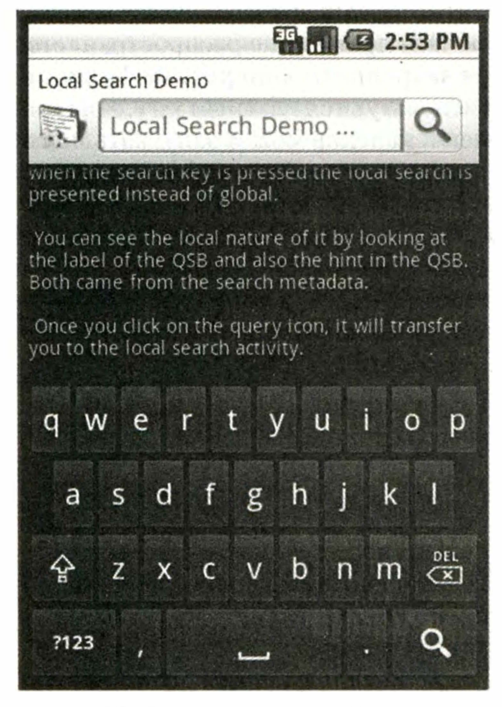


Рис. 14.21. QSB для локального поиска

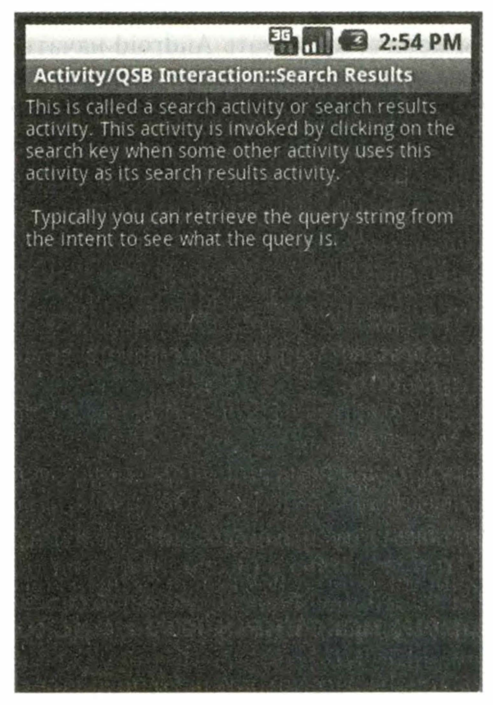


Рис. 14.22. Поисковые результаты, которые выдаются в ответ на запрос, введенный в поле для локального поиска

Хотя это явление и не использует для получения результатов текст поисковых запросов, оно позволяет понять, как определяется поисковое явление и как оно активируется. Ниже в этой главе мы покажем, как SearchActivity использует поисковые запросы, и рассмотрим различные действия, связанные с поиском, — явление должно на них отвечать.

Включение функции Type-to-Search

Пока мы исследовали несколько способов активации поиска, как локального, так и глобального. Мы показали, как осуществляется поиск при помощи поля QSB, расположенного на основном экране устройства. Мы рассказали, как активировать глобальный поиск из любого явления при условии, что явление допускает такой поиск. Для запуска поиска используется специальная экранная клавиша. Кроме того, было показано, как явление может запускать локальный поиск. Завершим эту

тему, изучив еще один способ активации поиска, называемый Type-to-Search (дословно «введите то, что нужно найти»).

Рассмотрев явление вроде RegularActivity, показанного на рис. 14.15, или LocalSearchActivity с рис. 14.20, вы увидите, что поиск запускается и при вводе в поле любой случайной буквы (например, t). Этот режим называется *Type-to-Search*, так как при нажатии любой клавиши, за которой в данном явлении не закреплено специальных функций, запустится поиск.

Назначение Type-to-Search несложно понять. Через любое имеющееся явление вы можете приказать Android начать поиск. Это правило не распространяется на клавиши, за которыми в данном явлении закреплены конкретные функции. Например, если для обслуживания явления используются клавиши x и y, а остальные клавиши специального значения не имеют, явление может запускать поиск при их нажатии, допустим при нажатии z или a. Этот режим удобен при работе с явлением, в котором уже отображены результаты поиска. В таком случае явление может интерпретировать нажатие клавиши как сигнал для запуска нового поиска.

Ниже показан код, используемый в методе onCreate() и запускающий такой режим работы (первый вариант — для запуска глобального поиска, второй — для локального):

```
this.setDefaultKeyMode(Activity.DEFAULT_KEYS_SEARCH_GLOBAL);
```

или

```
this.setDefaultKeyMode(Activity.DEFAULT_KEYS_SEARCH_LOCAL);
```

На этом мы завершаем обсуждение различных способов взаимодействия явлений Android и функции поиска. На примерах было продемонстрировано несколько способов активации локального и глобального поиска. Кроме того, мы коротко поговорили о поисковых явлениях и поставщиках поиска.

Далее мы продолжим исследование и реализуем простую программу-поставщик поиска.

Реализация простого поставщика поиска

Мы уже вкратце объяснили, как посредством поставщиков вариантов программы участвуют в глобальном поиске. Теперь рассмотрим этот вопрос подробнее.

Сначала объясним, как должна работать простая программа-поставщик поиска. Мы приведем список файлов, используемых при ее реализации, чтобы вы могли представить себе общую концепцию приложения, а также увидеть, из каких компонентов она будет состоять.

При написании поставщика поиска применяются два основных компонента. Первый отвечает за возвращение поисковых вариантов системе Android. Второй — это поисковое явление, принимающее запрос или вариант и превращающее эту информацию в результат поиска. Мы опишем «обязанности» каждого из этих компонентов и продемонстрируем, как они реализуются, на примерах исходного кода.

Кроме того, мы покажем, как создать простой поставщик поиска при помощи всего нескольких строк кода. Этот поставщик будет производным от готового поставщика SearchRecentSuggestionsProvider, который входит в состав в Android SDK. Затем рассмотрим, как простой поставщик поиска определяется в файле описания. Наконец, приведем используемые при этом методы SearchRecentSuggestionsProvider.

Мы покажем весь исходный код поискового явления и объясним, как оно определяется в файле описания. Кроме того, будет рассказано, как сохранять в поисковом явлении недавние запросы, чтобы они были доступны для использования простому поставщику поиска.

Мы также поговорим о поисковых метаданных, скрепляющих вместе поставщик поиска и поисковое явление, а потом внедрим в программу простое явление, инициирующее поиск, представляющее собой локальный механизм для запуска поиска.

Разговор завершим подробным рассмотрением программы. Так мы подготовимся к следующему разделу, где с нуля реализуем пользовательский поставщик поиска, не применяющий SearchRecentSuggestionsProvider.

Но сначала спланируем, как будет построен наш простой поставщик поиска.

Планирование простого поставщика поиска

Поскольку мы собираемся создать поставщик поиска, который будет наследовать свойства SearchRecentSuggestionsProvider, то в результате получим поставщик поиска с весьма ограниченным набором функций.

SearchRecentSuggestionsProvider позволяет сохранять запросы в том виде, в каком они представлены в поле QSB поискового явления. После того как они будут сохранены, система сможет отправлять их обратно в поле быстрого поиска в качестве подсказок; эта операция выполняется поставщиком поиска.

В производном поставщике поиска мы просто инициализируем базовый поставщик. Больше ничего здесь делать не требуется. Кроме того, присовокупим сюда самое простое поисковое явление (обычный текстовый вид), чтобы показать, что это поисковое явление активируется. В рамках поискового явления будут показаны методы, используемые при сохранении запросов.

Когда приложение будет готово, наша цель — добиться, чтобы вводившиеся ранее варианты запросов выводились в качестве вариантов в поле быстрого поиска.

Теперь рассмотрим список файлов, которые будут использоваться при реализации этого проекта.

Файлы для реализации простого поставщика поиска

Выше было сказано, что при реализации поставщика поиска основную роль играют два файла: SearchActivity.java и SimpleSuggestionProvider.java. Однако для завершения проекта нужны еще некоторые вспомогательные файлы. Мы перечислим все эти файлы и коротко опишем функции каждого. При объяснении выполненного нами решения мы приведем исходный код всех этих файлов.

Начнем с файлов Java:

- SimpleSuggestionProvider.java — реализует интересующий нас поставщик поиска;
- SearchActivity.java — необходим для работы с поставщиком поиска;
- SimpleMainActivity.java — необязательное явление, в котором демонстрируются локальные поисковые варианты.

Соответствующие файлы шаблонов:

- main.xml — файл шаблона для SimpleMainActivity;
- search_activity.xml — файл шаблона для SearchActivity.

Файл метаданных поиска: /xml/searchable.xml — здесь поисковое явление стыкуется с поставщиком поиска.

Разумеется, нам еще нужен файл описания manifest.xml — здесь в Android определяются компоненты приложения.

Рассмотрим все эти компоненты и начнем с реализации класса SimpleSuggestionProvider.

Реализация класса SimpleSuggestionProvider

В нашем проекте класс SimpleSuggestionProvider будет выполнять функции поставщика поиска, наследуя свойства SearchRecentSuggestionsProvider. Сначала узнаем, какие функции должен выполнять простой поставщик поиска.

Задачи простого поставщика поиска

Поскольку наш простой поставщик поиска является производным от SearchRecentSuggestionsProvider, большая часть задач выполняется базовым классом. Наш производный поставщик поиска должен инициализировать базовый класс при помощи имеющегося у него уникального источника. Поэтому в Android поставщик поиска активируется на основе уникального URI поставщика содержимого.

Подготовив поставщик поиска, мы должны конфигурировать его в файле описания как обычный поставщик содержимого, имеющий источник, а в XML-файле метаданных поиска — как поставщик поиска. Кроме того, в поисковых метаданных осуществляется соединение поставщика поиска с поисковым явлением.

Рассмотрим исходный код этого поставщика и узнаем, как выполняются вышеописанные требования.

Полный исходный код SimpleSuggestionProvider

Поскольку наследуются свойства SearchRecentSuggestionsProvider, исходный код простого поставщика поиска, приведенный в листинге 14.15, совсем не сложен.

Листинг 14.15. SimpleSuggestionProvider.java

```
// SimpleSuggestionProvider.java
public class SimpleSuggestionProvider
extends SearchRecentSuggestionsProvider {

   final static String AUTHORITY =
```

```
      "com.ai.android.search.simplesp.SimpleSuggestionProvider";
   final static int MODE = DATABASE_MODE_QUERIES;

   public SimpleSuggestionProvider() {
      super();
      setupSuggestions(AUTHORITY, MODE);
   }
}
```

В этом коде важно отметить несколько вещей. Первая — это строка с источником (authority string). Это тот самый источник, который мы рассматривали в главе 3 при изучении поставщика содержимого. Данная строка с источником должна быть уникальной и соответствовать собственному определению в файле описания (файл описания показан в листинге 14.16).

Кроме того, в коде инициализируется родительский класс, для этого применяется метод `setupSuggestions()`. Этот метод принимает два аргумента: первый — строка с источником (authority string), второй — так называемый режим базы данных (database mode).

Поговорим об этом режиме.

Понятие о режимах работы SearchRecentSuggestionsProvider

Основная функция входящего в состав Android `SearchRecentSuggestionsProvider` — сохранение запросов в базе данных, чтобы позже они могли использоваться в качестве вариантов в ходе живого поиска. С поисковым запросом ассоциируются две текстовые строки (см. рис. 14.3). Только первая строка является обязательной. При применении `SearchRecentSuggestionsProvider` для сохранения этих строк нужно сообщить системе, сколько строк требуется сохранить: одну или две.

Для выполнения этого требования базовый поставщик поиска в Android поддерживает два режима работы. В обоих режимах используется следующий префикс:

```
DATABASE_MODE_....
```

Вот оба режима:

```
DATABASE_MODE_QUERIES
DATABASE_MODE_2LINES
```

При применении первого режима сохраняется и при необходимости воспроизводится только одна строка. Во втором режиме указывается, что поставщик поиска может сохранять две строки. Первая строка — это запрос, а другая — строка описания (description line), выводящая на экран в ходе живого поиска элементы-варианты.

`SearchActivity` отвечает за сохранение этих вариантов, когда система вызывает его для отклика на запросы. `SearchActivity` вызовет следующий метод для сохранения этих элементов (данный метод более подробно обсуждается в разделе, рассказывающем о поисковом явлении):

```
public void saveRecentQuery (String queryString, String line2);
```

queryString — это строка (последовательность символов) в том виде, в котором ее ввел пользователь. Эта строка при живом поиске будет отображаться как один из вариантов, и если пользователь нажмет на вариант при поиске, вариант будет отправлен явлению, в котором происходит поиск (превратившись, таким образом, в новый поисковый запрос).

Вот что сказано в документации Android об аргументе line2.

Если вы конфигурировали ваш последний поставщик содержимого при помощи DATABASE_MODE_2LINES, здесь можно передать вторую строку текста. Она будет показана более мелким шрифтом под основным поисковым вариантом. При наборе совпадения с фрагментами из обеих строк будут отображаться в виде списка. Если вы не конфигурировали двухстрочный режим либо тот или иной поисковый запрос не имеет никакого дополнительного текста, здесь можно передать нулевое значение.

СОВЕТ

Более подробно этот готовый поставщик поиска описан по адресу http://developer.android.com/reference/android/provider/SearchRecentSuggestions.html.

Теперь, когда у нас есть исходный код для простого поставщика поиска, посмотрим, как этот поставщик регистрируется в файле описания.

Объявление поставщика поиска в файле описания

Поскольку SimpleSuggestionProvider является по существу поставщиком содержимого, он регистрируется в файле описания, как и любой другой поставщик содержимого. В листинге 14.16 показан файл описания. Важнейшие фрагменты выделены полужирным шрифтом.

Листинг 14.16. Файл описания SimpleSuggestionProvider

```
// имя файла: manifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.ai.android.search.simplesp"
      android:versionCode="1"
      android:versionName="1.0.0">
   <application android:icon="@drawable/icon"
         android:label="Simple Search Suggestion Provider:SSSP">
      <activity android:name=".SimpleMainActivity"
            android:label="SSSP:Simple Main Activity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>

<!--
*******************************************************************
* Код, относящийся к поиску: поисковое явление
*******************************************************************
```

```
-->
   <activity android:name=".SearchActivity"
            android:label="SSSP: Search Activity"
            android:launchMode="singleTop">
      <intent-filter>
         <action android:name="android.intent.action.SEARCH" />
         <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
      <meta-data android:name="android.app.searchable"
                        android:resource="@xml/searchable" />
   </activity>

   <meta-data android:name="android.app.default_searchable"
                        android:value=".SearchActivity" />

   <provider android:name=".SimpleSuggestionProvider"
      android:authorities
         ="com.ai.android.search.simplesp.SimpleSuggestionProvider" />
</application>
   <uses-sdk android:minSdkVersion="4" />
</manifest>
```

Обратите внимание: в исходном коде (см. листинг 14.15) и файле описания (см. листинг 14.16) источники (authority) совпадают. В обоих случаях значение источника:

```
com.ai.android.search.simplesp.SimpleSuggestionProvider
```

Мы рассмотрим и другие разделы файла описания при разговоре о прочих аспектах простого поставщика поиска.

Понятие о поисковом явлении простого поставщика поиска

Как уже было сказано выше, реализация поставщика поиска состоит из двух частей: самого поставщика и поискового явления, способного отвечать на предлагаемые поставщиком поисковые варианты. Мы уже описали порядок реализации поставщика поиска. Соответствующее поисковое явление будет рассмотрено ниже.

Следующий раздел будет построен примерно так же, как и предыдущий. Начнем с обсуждения основных функций поискового явления. Затем представим весь исходный код, чтобы вы в общей перспективе увидели, как выполняются эти функции.

Назначение простого поискового явления

Поисковое явление активируется встроенным в Android поисковиком при помощи строки запроса (query string). Поисковое явление, в свою очередь, должно считать эту строку запроса из намерения и сделать то, что необходимо.

Поисковое явление, как и любое явление, может активироваться различными намерениями и действиями. По этой причине рекомендуется проверять, в результате

действия какого намерения был активирован поиск. В нашем случае, когда явление активируется поисковым механизмом Android, соответствующее действие называется ACTION_SEARCH.

При определенных обстоятельствах поисковое явление может активировать само себя. Если вы собираетесь использовать такой метод, то режим запуска поискового явления следует определить как singleTop. Кроме того, явление должно будет запускать метод onNewIntent(). Этот аспект также будет рассмотрен отдельно — в подразделе «OnCreate() и onNewIntent()» далее.

Собираясь осуществлять какие-либо операции со строкой запроса, мы просто занесем ее в журнал (лог). После того как запрос будет зарегистрирован таким образом, нам понадобится сохранить его в SearchRecentSuggestionsProvider — так, чтобы этот вариант при необходимости предоставлялся и при будущих операциях поиска.

Теперь рассмотрим исходный код данного класса поискового явления.

Полный исходный код поискового явления

В листинге 14.17 показан исходный код обсуждаемого здесь класса SearchActivity.

Листинг 14.17. Поисковое явление SimpleSuggestionProvider

```
// имя файла: SearchActivity.java
public class SearchActivity extends Activity
{
   private final static String tag ="SearchActivity";
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      Log.d(tag,"I am being created");
      // в противном случае выполнить
      setContentView(R.layout.layout_test_search_activity);
      // this.setDefaultKeyMode(Activity.DEFAULT_KEYS_SEARCH_GLOBAL);
      this.setDefaultKeyMode(Activity.DEFAULT_KEYS_SEARCH_LOCAL);

      // получение и обработка поискового запроса происходит здесь
      final Intent queryIntent = getIntent();
      final String queryAction = queryIntent.getAction();
      if (Intent.ACTION_SEARCH.equals(queryAction))
      {
         Log.d(tag,"new intent for search");
         this.doSearchQuery(queryIntent);
      }
      else {
         Log.d(tag,"new intent NOT for search");
      }
      return;
   }

   @Override
   public void onNewIntent(final Intent newIntent)
   {
```

```
        super.onNewIntent(newIntent);
        Log.d(tag,"new intent calling me");

        // получение и обработка поискового запроса происходит здесь
        final Intent queryIntent = getIntent();
        final String queryAction = queryIntent.getAction();
        if (Intent.ACTION_SEARCH.equals(queryAction))
        {
            this.doSearchQuery(queryIntent);
            Log.d(tag,"new intent for search");
        }
        else {
            Log.d(tag,"new intent NOT for search");
        }
    }
    private void doSearchQuery(final Intent queryIntent)
    {
        final String queryString =
            queryIntent.getStringExtra(SearchManager.QUERY);

        // запись строки поискового запроса
        // в поставщик недавно предлагавшихся вариантов
        SearchRecentSuggestions suggestions =
                new SearchRecentSuggestions(this,
            SimpleSuggestionProvider.AUTHORITY,
            SimpleSuggestionProvider.MODE);
        suggestions.saveRecentQuery(queryString, null);
    }
}
```

Рассмотрим, как поисковое явление проверяет, какое действие было выполнено, и находит строку поискового запроса.

Определение выполненного действия и выполнение запроса

В коде поискового явления осуществляется проверка, каким действием было запущено это явление. Это делается путем просмотра инициирующего намерения и сравнения его с constant intent.ACTION_SEARCH. Если регистрируется совпадение, то вслед за этим активируется функция doSearchQuery().

При выполнении функции doSearchQuery() поисковое явление находит поисковый запрос при помощи намерения extra. Код таков:

```
final String queryString =
    queryIntent.getStringExtra(SearchManager.QUERY);
```

Обратите внимание: это намерение определяется как SearchManager.QUERY. По ходу этой главы мы сталкивались с множеством таких намерений, все они описаны в справке по API SearchManager. (В разделе «Ресурсы» в конце этой главы приведена URL этого справочного документа.)

OnCreate() и onNewIntent()

Android начинает работу с поисковым явлением, когда пользователь вводит текст в окно для поиска и нажимает либо один из вариантов, либо символ лупы. Так создается поисковое явление и вызывается его метод onCreate(). Намерение, передаваемое методу onCreate(), будет иметь набор действий ACTION_SEARCH.

В некоторых случаях поисковое явление не создается, а вместо этого при помощи метода onNewIntent() сообщаются новые поисковые критерии. Как это происходит? Метод обратного вызова onNewIntent() тесно связан с типом запуска (launching mode) явления. При внимательном рассмотрении листинга 14.16 видно, что в файле описания поисковое явление определено как singleTop.

Когда поисковое явление определяется как singleTop, оно приказывает Android не создавать нового явления, если singleTop уже находится на верхней позиции в стеке. В таком случае Android вызывает не onCreate(), а onNewIntent(). Вот почему в исходном коде явления (см. листинг 14.17) намерение проверяется в двух местах.

Как тестировать onNewIntent()

Когда onNewIntent() будет реализован, вы заметите, что он не активируется в обычном порядке. Возникает вопрос: когда же поисковое явление оказывается на верхней позиции в стеке? Обычно этого не происходит.

Этого не случается по следующей причине. Допустим, явление A активирует поиск, запуская в результате поисковое явление B. Затем явление B отображает результаты, и для возврата в предыдущее окно пользователь применяет кнопку **Back** (Назад), а в этот момент явление B, то есть актуальное поисковое явление, покидает верхнюю позицию в стеке, уступая ее явлению A. Либо пользователь может нажать клавишу возврата на домашнюю страницу и воспользоваться глобальным поиском на домашнем экране, в результате чего на верхней позиции в стеке окажется явление основного экрана.

Единственный способ работы, гарантирующий, что поисковое явление окажется на верхней позиции в стеке, таков: допустим, в ходе поиска явление A запускает явление B. Если явление B определяет Type-to-Search, то, когда явление B находится в фокусе, поиск вновь активирует явление B, но уже с новыми критериями. В листинге 14.17 показано, как мы настроили функцию Type-to-Search в качестве примера. Вновь обратимся к коду:

```
this.setDefaultKeyMode(Activity.DEFAULT_KEYS_SEARCH_LOCAL);
```

Сохранение запроса при помощи SearchRecentSuggestionsProvider

Итак, мы обсудили, как поисковое явление должно сохранять встреченные запросы, чтобы позже они могли предлагаться в качестве вариантов при живом поиске. Ниже приведен фрагмент кода, в котором выполняется эта задача.

```
final String queryString =
   queryIntent.getStringExtra(SearchManager.QUERY);
// Запись строки поискового запроса в поставщик
```

```
// недавно предлагавшихся вариантов.
SearchRecentSuggestions suggestions = new SearchRecentSuggestions(this,
   SimpleSuggestionProvider.AUTHORITY,
   SimpleSuggestionProvider.MODE);
suggestions.saveRecentQuery(queryString, null);
```

Этот код иллюстрирует одно из сделанных выше замечаний: Android передает информацию запроса ДОПОЛНИТЕЛЬНО вместе с намерением.

Когда запрос будет доступен, можно приказать базовому SearchRecentSuggestions-Provider сохранить его, инстанцировав новый объект-вариант и выполнив операцию сохранения. Поскольку мы использовали однострочный режим, второй аргумент saveRecentQuery равен нулю.

Теперь рассмотрим определение метаданных поиска, тот элемент, который связывает поисковое явление с поставщиком поиска.

Метаданные поиска

Определение поиска в Android начинается с поискового явления. Сначала оно происходит в файле описания. В рамках этого определения вы сообщаете Android, где находится XML-файл с поисковыми метаданными (см. листинг 14.16).

В листинге 14.18 показан файл поисковых метаданных нашего приложения.

Листинг 14.18. Поисковые метаданные SimpleSuggestionProvider

```
// имя файла: searchable.xml
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
   android:label="@string/search_label"
   android:hint="@string/search_hint"
   android:searchMode="showSearchLabelAsBadge"

   android:includeInGlobalSearch="true"
   android:searchSuggestAuthority=
         "com.ai.android.search.simplesp.SimpleSuggestionProvider"
   android:searchSuggestSelection=" ? "
/>
```

В этом листинге присутствуют три атрибута, важных для работы поставщика поиска. Рассмотрим их по отдельности.

Первый атрибут — includeInGlobalSearch — приказывает Android использовать этот поставщик поиска как один из источников при глобальном быстром поиске.

Второй атрибут — searchSuggestAuthority — указывает на источник поставщика поиска в том виде, в котором источник дан в файле описания (см. листинг 14.16).

Третий атрибут — searchSuggestSelection — всегда имеет значение ? при условии, что он является производным от поставщика недавно предложенных вариантов. Строка передается поставщику поиска как строка selection метода query, относящегося к поставщику содержимого. Обычно эта строка представляет собой условие where, входящее в состав оператора выбора (selection statement). Затем Android передает запрос как первую запись в массиве выбранных аргументов (select argument array) метода query поставщика содержимого. Поскольку код для реагирования

на эти нюансы скрыт в поставщике недавно предложенных вариантов, мы не сможем показать, как эти аргументы используются с методом query поставщика содержимого. Этот вопрос будет более подробно рассмотрен далее.

На этом мы завершаем разговор о написании поискового явления для простого поставщика поиска. Теперь, когда мы поговорили и о поставщике вариантов, и о поисковом явлении, обсудим инициирующее явление (invoker activity), которое будет играть в нашем приложении роль основной входной точки и при помощи которого мы сможем протестировать локальный поиск.

Явление инициирования поиска (search invoker)

Хотя это явление и не требуется нам для завершения поставщика поиска, оно, находясь в фокусе, позволит инициировать локальный поиск. В листинге 14.19 показан исходный код явления, активирующего поиск.

Листинг 14.19. SimpleSuggestionProvider: основное явление

```
public class SimpleMainActivity extends Activity
{
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
   }
}
```

Изучив определение данного явления в файле описания (см. листинг 14.16), можно отметить, что явление SearchActivity прямо не указано как заданное по умолчанию для локального поиска. Это объясняется тем, что мы использовали данную спецификацию на уровне приложения, а не на уровне явления. Чтобы применить спецификацию, в файле описания нужны две следующие строки кода:

```
<meta-data android:name="android.app.default_searchable"
        android:value=".SearchActivity" />
```

Обратите внимание — в файле описания две эти строки не входят в состав ни одного явления (см. листинг 14.16). Спецификация указывает Android, что все явления данного приложения используют явление SearchActivity по умолчанию. В том числе это касается самого SearchActivity. Этим фактом вы можете воспользоваться при активации onNewIntent(), нажимая поисковую клавишу при проверке результатов SearchActivity. Но этого не произойдет, если вы собираетесь по умолчанию задать способ поиска только для инициирующего явления, а не для целого приложения.

Ниже приведен простой шаблон, используемый с явлением, инициирующим основной поиск:

```
// имя файла: /res/layout/main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/text1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/main_activity_text"
    />
</LinearLayout>
```

Рассмотрим файл strings.xml, используемый с этим файлом шаблона и с остальной частью программы:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="main_activity_text">
    Это простое явление. Нажмите поисковую клавишу
    для активации локального поиска
    \n\n
    Поставщик поиска также будет участвовать
    в глобальном поиске. Если вы попадете к этому
    приложению через глобальный поиск,
    то не увидите этот вид, а непосредственно
    перейдете к виду searchactivity.
    </string>

    <string name="search_activity_text">
    Если вы видите это явление, то попали сюда
    либо через глобальный, либо через
    локальный поиск.
    \n\n
    Это явление также активирует Type-to-Search. Здесь также
    демонстрируются концепции singletop/new intent.
    </string>

    <string name="app_name">Simple Suggestion Provider</string>
    <string name="search_label">Local Search Demo</string>
  <string name="search_hint">Local Search Hint</string>
</resources>
```

Опыт работы пользователей с простым поставщиком поиска

При запуске этой программы вы видите основной экран, который будет выглядеть как на рис. 14.23 (это наше явление, инициирующее поиск).

Если нажать поисковую клавишу, когда это явление находится в фокусе, то активируется локальный поиск (рис. 14.24).

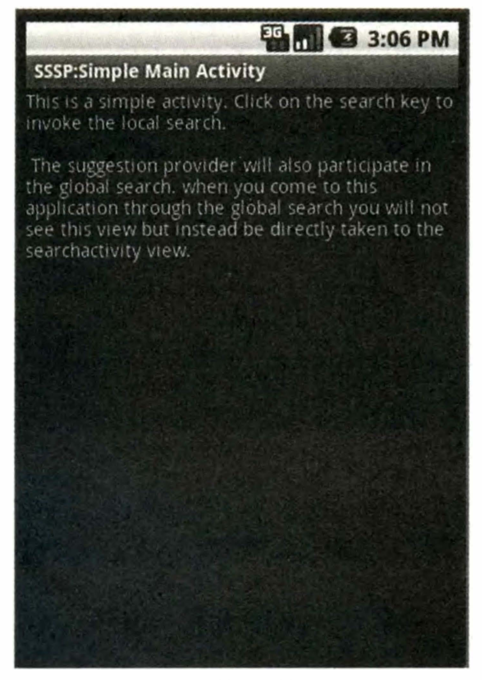


Рис. 14.23. Простой поставщик поиска: основное явление (включено для локального поиска)



Рис. 14.24. Простой поставщик поиска: поле для локального быстрого поиска

Как видите, на рис. 14.24 отсутствуют поисковые варианты, так как мы пока еще ничего не искали. Кроме того, из рисунка понятно, что мы имеем дело с локальным поиском; метка и подсказка в поисковом поле — те самые, которые мы задали в XML-файле поисковых метаданных.

Начнем поиск и найдем строку test1. Сначала мы перейдем на экран поискового явления (search activity), показанный на рис. 14.25.

Из исходного кода SearchActivity, показанного в листинге 14.17, мы видим, что это явление не выполняет на экране ничего эффектного — лишь «за кулисами» оно сохраняет строки запросов в базе данных. Теперь, если вы перейдете обратно к основному экрану (нажав кнопку Back (Назад)) и вновь активируете поиск, то увидите следующий экран (рис. 14.26), на котором в качестве вариантов живого поиска предлагаются вводившиеся ранее строки поисковых запросов.

Сейчас самое время рассмотреть, как активируется onNewIntent(). Если вы работаете с поисковым явлением (см. рис. 14.24), можете ввести букву, например, t — и система снова активирует поиск методом Type-to-Search. Тогда вы увидите, что в журнале отладки отмечен вызов onNewIntent().

Рассмотрим, что нужно сделать, чтобы эти варианты отображались в поле для быстрого глобального поиска. Поскольку мы активировали includeInGlobalSearch, варианты должны отображаться и в глобальном QSB. Но прежде, чем вы сможете отображать варианты в поле для глобального поиска, в программе нужно будет активировать возможность показа вариантов глобального поиска (рис. 14.27).

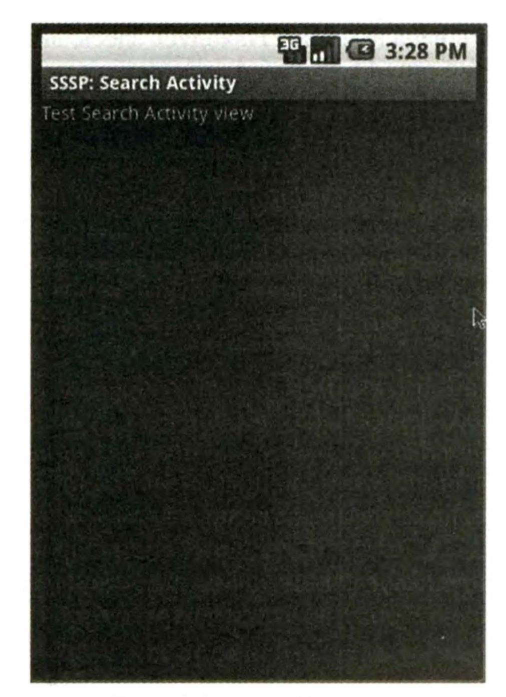


Рис. 14.25. Простой поставщик поиска: поле для отображения результатов локального поиска



Рис. 14.26. Простой поставщик поиска: найденный вариант локального поиска

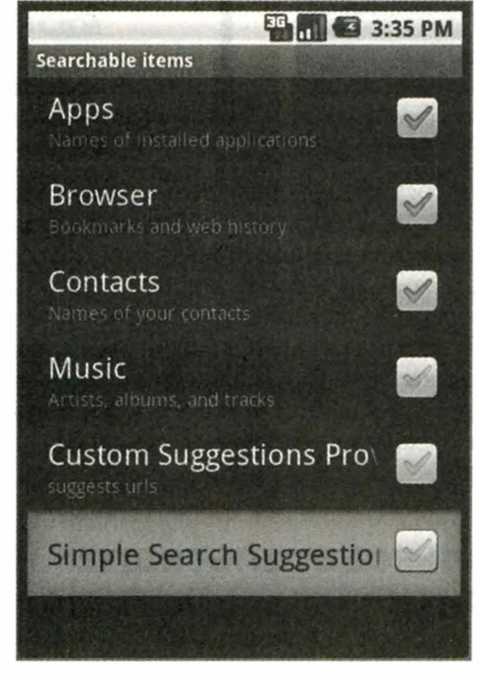
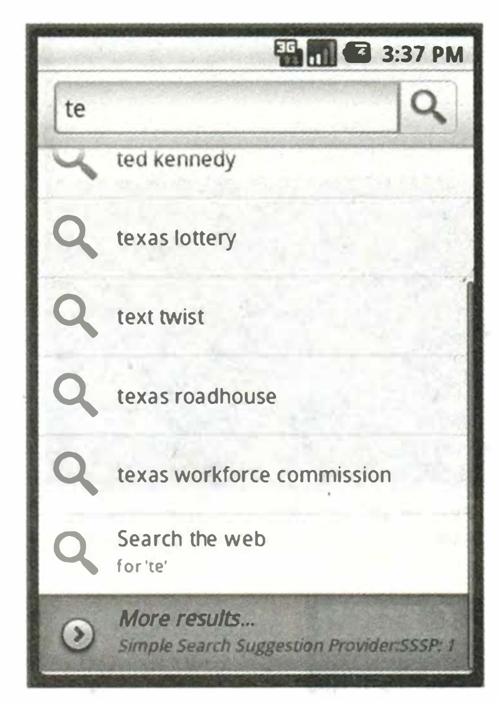


Рис. 14.27. Активация простого поставщика поиска

Как попасть на этот экран, мы рассказали в начале главы. Находясь здесь, вы увидите, как глобальный поиск, показанный на рис. 14.28, работает с нашим поставщиком поиска.

При навигации по глобальному поиску к определенному элементу вы увидите окно локального поиска (рис. 14.29).



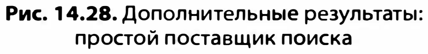


Рис. 14.28. Дополнительные результаты: простой поставщик поиска

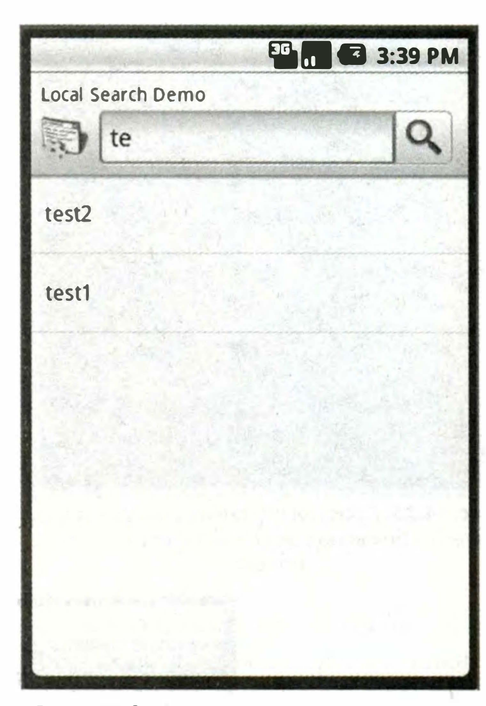


Рис. 14.29. Переход к локальному поиску: простой поставщик поиска

На этом мы завершаем обсуждение простого поставщика поиска. Вы научились использовать встроенный `RecentSearchSuggestionProvider`, который запоминает поисковые вопросы, специфичные для данного приложения. Такой метод позволяет, написав минимальный объем кода, принимать запросы, введенные в ходе локального поиска, и предлагать их в качестве вариантов даже в контексте глобального поиска.

Правда, в этом простом упражнении мы не рассмотрели, как написать поставщик поиска с нуля. Гораздо важнее, что мы помогли вам понять, как поставщик поиска возвращает набор вариантов и какие колонки имеются в этом наборе вариантов. Чтобы лучше понять этот механизм, реализуем с нуля собственный (пользовательский) поставщик поиска.

Реализация пользовательского поставщика поиска

Поиск в Android настолько гибок, что так и хочется заняться его настройкой. Поскольку в предыдущем разделе мы использовали готовый поставщик поиска,

многие его функции остались скрыты в SearchRecentSuggestionsProvider и не рассмотрены. Далее мы подробно рассмотрим, как строится поставщик поиска, и для этого внедрим пользовательский поставщик, который назовем SuggestUrlProvider.

Сначала объясним, как должен работать этот SuggestUrlProvider. Затем приведем список файлов, используемых при реализации поставщика поиска. Эти файлы должны дать общее представление о том, как строится пользовательский поставщик поиска.

Как уже было указано выше, при написании поставщика поиска реализуется два основных компонента: сам поставщик поиска и соответствующее поисковое явление. Как и в случае реализации простого поставщика поиска, мы рассмотрим оба этих компонента в контексте их функций и способов осуществления этих функций.

При реализации поставщика поиска будем обращать внимание на типы URL, применяемых при активации поставщика поиска, на то, как поставщику поиска передается неполный (partial) текст, рассмотрим список колонок, возвращаемый поставщиком содержимого, а также узнаем, как поставщик поиска передает информацию поисковому явлению.

При реализации поискового явления выясним, как активируется это явление и как сообщаются действия, выполняемые в ходе поиска. Мы покажем, как поисковое явление может получать значения из намерения, используемого для активации явления.

Наконец, вы узнаете, как работать с готовым приложением.

Планирование пользовательского поставщика поиска

Наш поставщик поиска будет называться SuggestURLProvider. Цель этого поставщика — отслеживать, какие запросы пользователь вводит в поле для быстрого поиска. Если поисковый запрос содержит текст вида great.m (в данном случае окончание .m соответствует meaning, то есть «значение»), поставщик интерпретирует первую часть запроса как слова и предложит URL из Интернета, которую можно открыть в ответ на этот запрос.

Для каждого слова наш поставщик предложит две URL. Первая URL позволяет пользователю найти слово на сайте http://www.thefreedictionary.com, а вторая URL — найти слово в Google. При выборе одного из вариантов пользователь попадает прямо на первый или на второй сайт. Если пользователь нажмет поисковую пиктограмму в поле QSB, то поисковое явление просто зарегистрирует текст этого запроса в простом шаблоне данного явления. Чтобы яснее представить этот процесс, мы покажем снимки экранов, на которых происходит такое взаимодействие.

Рассмотрим список файлов, входящих в состав данного решения (проекта).

Файлы для реализации проекта SuggestURLProvider

Мы уже упоминали выше, что два основных файла — это SearchActivity.java и SuggestUrlProvider.java. Однако для завершения проекта также понадобятся

вспомогательные файлы. Ниже приводится список этих файлов и краткое описание функции каждого. В процессе мы покажем исходный код каждого из этих файлов.

- SuggestUrlProvider.java — в данном файле реализуется протокол пользовательского поставщика поиска. В таком случае пользовательский поставщик поиска интерпретирует поисковые строки как слова и возвращает пару вариантов при помощи специального курсора (suggestions cursor).
- SearchActivity.java — это явление отвечает за получение запросов или вариантов, предоставляемых поставщиком поиска. Определение SearchActivity также обеспечивает связывание поставщика поиска с данным явлением.
- layout/layout_search_activity.xml — этот файл шаблона может использоваться с SearchActivity. В нашем примере именно в этом файле мы осуществляем регистрацию присылаемых запросов.
- values/strings.xml — содержит определения строк для шаблона, заголовка локального поиска, подсказки в поле локального поиска и т. д.
- xml/searchable.xml — XML-файл метаданных поиска, скрепляющий вместе SearchActivity, поставщик поиска и поле QSB.
- manifest.xml — файл описания программы, здесь определяются поисковое явление и поставщик поиска. Кроме того, здесь задается SearchActivity, которое следует активировать для локального поиска по данной программе.

В этом списке главную роль играют файлы SuggestUrlProvider и SearchActivity. Начнем с изучения SuggestUrlProvider.

Реализация класса SuggestUrlProvider

При создании пользовательского поставщика поиска мы используем класс SuggestUrlProvider. Он участвует в реализации протокола поставщика поиска. Начнем изучение SuggestUrlProvider с выполняемых им функций.

Функции поставщика поиска

Поставщик поиска активируется поисковым механизмом Android посредством URI, идентифицирующего поставщик, и дополнительного аргумента, в котором передается запрос.

При поиске в Android для активации поставщика вариантов применяются URI двух типов. Первый идентификатор называется «поисковый URI». Он используется для сбора вариантов. Ответ должен состоять из одной или более строк и содержать заранее известный набор колонок.

Второй URI называется «URI варианта» и используется для предложения вариантов, уже хранящихся в кэше устройства. Ответ должен состоять из одной или более строк и содержать заранее известный набор колонок.

Кроме того, поставщик поиска должен содержать в файле поисковых метаданных указание о том, каким образом будет происходить получение неполных поисковых запросов. Это может выполняться при помощи аргумента select метода query или в последнем сегменте пути в самом URI (URI также передается как один из аргументов методу запроса, применяемому поставщиком).

В поставщике поиска содержатся колонки, каждая из которых запускает определенный механизм поиска. Сначала поставщик поиска должен решить относительно набора колонок, который должен быть возвращен, следующие вопросы:

- одна из колонок должна использоваться для кэширования вариантов, возвращаемых при поиске Android;
- колонки должны содержать алгоритм решения о том, должен ли поисковый запрос заменять текст, находящийся в поле запроса, а также должны контролировать этот процесс;
- колонки должны использоваться, когда в результате поиска необходимо непосредственно активировать действие, а не отображать список результатов после того, как пользователь щелкнет на одном из вариантов, предложенных в ходе живого поиска.

Весь исходный код SuggestUrlProvider

В листинге 14.20 показан исходный код класса SuggestUrlProvider. Далее в главе отдельные разделы этого листинга будут рассмотрены в деталях по мере подробного изучения отдельных функций данного кода.

Листинг 14.20. Исходный код CustomSuggestionProvider

```
public class SuggestUrlProvider extends ContentProvider
{
   private static final String tag = "SuggestUrlProvider";
   public static String AUTHORITY =
      "com.ai.android.search.custom.suggesturlprovider";

   private static final int SEARCH_SUGGEST = 0;
   private static final int SHORTCUT_REFRESH = 1;
   private static final UriMatcher sURIMatcher = buildUriMatcher();

   private static final String[] COLUMNS = {
         "_id", // эта колонка является обязательной
         SearchManager.SUGGEST_COLUMN_TEXT_1,
         SearchManager.SUGGEST_COLUMN_TEXT_2,
         SearchManager.SUGGEST_COLUMN_INTENT_DATA,
         SearchManager.SUGGEST_COLUMN_INTENT_ACTION,
         SearchManager.SUGGEST_COLUMN_SHORTCUT_ID
         };

   private static UriMatcher buildUriMatcher()
   {
      UriMatcher matcher = new UriMatcher(UriMatcher.NO_MATCH);
      matcher.addURI(AUTHORITY,
         SearchManager.SUGGEST_URI_PATH_QUERY,
         SEARCH_SUGGEST);
      matcher.addURI(AUTHORITY,
         SearchManager.SUGGEST_URI_PATH_QUERY +
         "/*",
         SEARCH_SUGGEST);
      matcher.addURI(AUTHORITY,
```

```
            SearchManager.SUGGEST_URI_PATH_SHORTCUT,
            SHORTCUT_REFRESH);
        matcher.addURI(AUTHORITY,
            SearchManager.SUGGEST_URI_PATH_SHORTCUT +
            "/*",
            SHORTCUT_REFRESH);
        return matcher;
    }

    @Override
    public boolean onCreate() {
        // не позволяет больше ничего делать, в частности
        Log.d(tag,"onCreate called");
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder)
    {
        Log.d(tag,"query called with uri:" + uri);
        Log.d(tag,"selection:" + selection);

        String query = selectionArgs[0];
        Log.d(tag,"query:" + query);

        switch (sURIMatcher.match(uri)) {
            case SEARCH_SUGGEST:
                Log.d(tag,"search suggest called");
                return getSuggestions(query);
            case SHORTCUT_REFRESH:
                Log.d(tag,"shortcut refresh called");
                return null;
            default:
                throw new IllegalArgumentException("Unknown URL " + uri);
        }
    }

    private Cursor getSuggestions(String query)
    {
        if (query == null) return null;
        String word = getWord(query);
        if (word == null)
            return null;

        Log.d(tag,"query is longer than 3 letters");

            MatrixCursor cursor = new MatrixCursor(COLUMNS);
            // cursor.addRow(createRow(query,"rowl"));
            cursor.addRow(createRow1(word));
            cursor.addRow(createRow2(word));
            return cursor;
```

```
}
private Object[] createRow1(String query)
{
   return columnValuesOfQuery(query,
      "android.intent.action.VIEW",
      "http://www.thefreedictionary.com/" + query,
      "Look up in freedictionary.com for",
      query);
}

private Object[] createRow2(String query)
{
   return columnValuesOfQuery(query,
      "android.intent.action.VIEW",
      "http://www.google.com/search?hl=en&source=hp&q=define%3A/"
      + query,
      "Look up in google.com for",
      query);
}
private Object[] columnValuesOfQuery(String query,
      String intentAction,
      String url,
      String text1,
      String text2)
{
   return new String[] {
         query,          // _id
         text1,          // текст1
         text2,          // текст2
         url,         // intent_data
                      // (включается при нажатии элемента)
         intentAction,   //действие
         SearchManager.SUGGEST_NEVER_MAKE_SHORTCUT
   };
}

private Cursor refreshShortcut(String shortcutId, String[] projection) {
   return null;
}

public String getType(Uri uri) {
   switch (sURIMatcher.match(uri)) {
      case SEARCH_SUGGEST:
         return SearchManager.SUGGEST_MIME_TYPE;
      case SHORTCUT_REFRESH:
         return SearchManager.SHORTCUT_MIME_TYPE;
      default:
         throw new IllegalArgumentException("Unknown URL " + uri);
   }
}

public Uri insert(Uri uri, ContentValues values) {
```

```
      throw new UnsupportedOperationException();
   }

   public int delete(Uri uri, String selection, String[] selectionArgs) {
      throw new UnsupportedOperationException();
   }

   public int update(Uri uri, ContentValues values, String selection,
         String[] selectionArgs) {
      throw new UnsupportedOperationException();
   }

   private String getWord(String query)
 {
     int dotIndex = query.indexOf('.');
     if (dotIndex < 0)
         return null;
     return query.substring(0,dotIndex);
 }
}
```

Виды URI поставщика поиска

Теперь, когда мы рассмотрели весь исходный код пользовательского поставщика поиска, разберем, как в отдельных фрагментах этого кода выполняются конкретные функции URI.

Сначала рассмотрим формат того URI, который Android использует для активации поставщика поиска. Если поставщик поиска имеет источник:

```
com.ai.android.search.custom.suggesturlprovider
```

то Android отправляет два возможных URI. URI первого типа называется поисковым. Он может выглядеть так:

```
content://com.ai.android.search.suggesturlprovider/search_suggest_query
```

или так:

```
content://com.ai.android.search.suggesturlprovider/search_suggest_query/
     <your-query>
```

Данный URI выпускается после того, как пользователь начнет вводить текст в поле QSB. В одном из случаев запрос отправляется как дополнительный элемент в конце URI и является сегментом пути (path segment). В файле поисковых метаданных `searchable.xml` указывается, будет запрос посылаться как один из сегментов пути или нет. Эту спецификацию мы подробнее рассмотрим в разделе, посвященном поисковым метаданным.

URI второго типа, используемые с поставщиком поиска, относятся к применяемым в Android быстрым вариантам запроса (search shortcuts). Быстрый вариант запроса — это вариант (см. рис. 14.3), который Android кэширует, не требуя при этом от поставщика поиска нового контента. Подробнее быстрые варианты запросов в Android будут рассмотрены в разделе, посвященном изучению колонок

поискового варианта. Пока остановимся на том, что URI второго типа могут выглядеть так:

```
content://com.ai.android.search.suggesturlprovider/search_suggest_shortcut
```

или

```
content://com.ai.android.search.suggesturlprovider/search_suggest_shortcut/
   <shortcut-id>
```

Android выпускает такой URI в том случае, когда системе требуется определить, актуальны ли еще быстрые варианты запросов, сохраненные на данный момент в кэше. Такие URI называются URI быстрого доступа. Если поставщик возвращает только одну строку, он заменит имеющийся быстрый вариант на новый. Если поставщик посылает нулевое значение, то Android расценивает это как признак того, что имеющийся в кэше вариант не подходит.

В классе Android SearchManager определяются две константы, используемые для представления тех сегментов, которыми различаются URI двух типов (search_suggest_search и search_suggest_shortcut). Эти константы соответственно:

```
SearchManager.SUGGEST_URI_PATH_QUERY
SearchManager.SUGGEST_URI_PATH_SHORTCUT
```

Поставщик поиска должен распознавать типы таких входящих URI в своем методе query(). В листинге 14.20 показано, как эту задачу выполняет UriMatcher (подробнее об UriMatcher можно узнать из главы 3).

Реализация getType() и указание типов MIME

Поскольку поставщик поиска в конечном итоге является поставщиком содержимого, он должен реализовать соответствующий контракт, в рамках которого реализуется метод getType().

Можете вновь обратиться к листингу 14.20, где показано, как в таком случае реализуется getType(). В поисковом фреймворке Android в классе SearchManager предоставляется пара констант, обеспечивающих работу с типами MIME. В данном случае используется два типа MIME:

```
SearchManager.SUGGEST_MIME_TYPE
SearchManager.SHORTCUT_MIME_TYPE
```

Они означают:

```
vnd.android.cursor.dir/vnd.android.search.suggest
vnd.android.cursor.item/vnd.android.search.suggest
```

Передача запроса поставщику поиска: аргумент Selection

Когда Android использует для вызова поставщика URI одного из описанных выше типов, для получения курсора варианта осуществляется вызов метода query(), относящегося к поставщику поиска. Изучив реализацию метода query() в листинге 14.20, вы заметите, что с ним используются аргументы selection и selectionArgs, предназначенные для оформления и возвращения курсора.

Чтобы понять, какая информация передается в двух этих аргументах, просмотрите файл searchable.xml. Код этого файла метаданных приведен в листинге 14.21.

Листинг 14.21. Поисковые метаданные поставщика CustomSuggestionProvider

```
// xml/searchable.xml
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
    android:label="@string/search_label"
    android:hint="@string/search_hint"
    android:searchMode="showSearchLabelAsBadge"
    android:searchSettingsDescription="suggests urls"
    android:includeInGlobalSearch="true"

    android:searchSuggestAuthority=
            "com.ai.android.search.custom.suggesturlprovider"
    android:searchSuggestIntentAction="android.intent.action.VIEW"
    android:searchSuggestSelection=" ? "
/>
```

Обратите внимание на атрибут searchSuggestSelection в этом листинге, определяющем метаданные поиска. Он точно соответствует аргументу selection метода query() поставщика поиска. В главе 3 говорилось о том, что этот аргумент обычно используется для передачи условия where с заменяемыми символами ?. Затем массив заменяемых значений передается при помощи аргумента-массива selectionArgs. Именно это здесь и происходит. Если вы указываете searchSuggestSelection, Android предполагает, что вы хотите получить поисковый текст не через URI, а через аргумент selection метода query(). В таком случае поисковый механизм Android пошлет ? (обратите внимание на то, что перед символом вопроса и после него находятся пробелы) как значение аргумента selection, причем текст запроса является первым элементом в аргументе-массиве selection.

Если не указать searchSuggestSelection, то поисковый запрос будет передаваться как последний сегмент URI. Можете выбрать любой вариант. В нашем примере мы работали с selection, а не с URI.

Изучение поисковых метаданных пользовательского поставщика поиска

В этом пункте мы исследуем тему метаданных поиска и в процессе посмотрим, какие еще атрибуты могут использоваться в данном контексте. Мы обсудим в основном такие атрибуты, которые часто применяются либо имеют отношение к поставщикам поиска. Весь список приводится на странице API SearchManager: **http://developer.android.com/reference/android/app/SearchManager.html**.

Атрибут searchSuggestIntentAction используется для передачи или задания действия, осуществляемого в том случае, когда SearchActivity активируется намерением. Это позволяет SearchActivity осуществлять не только стандартный поиск, но и другие операции. В листинге 14.23 показано, как явление searchActivity ищет действие VIEW или SEARCH, проверяя значение действия, запускаемого намерением.

Еще один атрибут (в данном случае мы его не используем, но он применяется с поставщиками поиска) называется searchSuggestPath. Если он указывается, это

строковое значение прикрепляется к URI (тому, который активирует поставщик поиска) после SUGGEST_URI_PATH_QUERY.

Таким образом, отдельно взятый пользовательский поставщик поиска может реагировать на два разных поисковых явления. Каждое явление SearchActivity будет использовать собственное окончание URI.

Вы можете указать не только действие, запускаемое намерением, но и данные для намерения — для этого применяется атрибут searchSuggestIntentData. Это URI данных, который может передаваться поисковому явлению вместе с действием как часть намерения в момент активации.

Атрибут, который называется searchSuggestThreshold, задает количество символов, после ввода которого в поле для быстрого поиска активируется поставщик поиска. По умолчанию пороговое значение равно нулю.

Атрибут queryAfterZeroResults (может иметь значения true или false) указывает, следует ли связаться с поставщиком, если конкретный набор символов возвратит нулевое количество результатов для следующего набора символов.

Теперь, когда мы разобрались с URI, аргументами выбора и поисковыми метаданными, переходим к наиболее важному аспекту поставщика поиска: поисковому курсору.

Колонки поискового курсора

Поисковый курсор — это прежде всего курсор. Он мало отличается от курсора базы данных, который подробно рассматривался в главе 3. Поисковый курсор играет роль контракта между поисковым механизмом Android и поставщиком поиска. Это означает, что названия и типы колонок, возвращаемых в курсоре, жестко заданы и известны обеим сторонам.

Для обеспечения гибкого поиска поисковая функция в Android предусматривает много колонок, большинство из которых не являются обязательными. Поставщик содержимого может возвращать не все эти колонки. Можно игнорировать отсылку колонок, нерелевантных для конкретного поставщика поиска. В этом разделе будут рассмотрены значение и степень важности большинства колонок (все остальные колонки описаны на странице API SearchManager, уже упоминавшейся выше).

Сначала изучим колонки, которые может возвращать поставщик поиска, разберем значение каждой из колонок и выясним, как та или иная колонка воздействует на поиск.

Как и все курсоры, курсор поиска имеет колонку _id. Она обязательно должна присутствовать. Названия всех остальных колонок начинаются с префикса SUGGEST_COLUMN_. Эти названия — константы, они определяются в справке, описывающей API SearchManager. Ниже будут рассмотрены колонки, используемые чаще всего. Полный список дается в справке по API, ссылка на которую указана в конце этой главы.

- text_1 — это первая строка текста в поисковом варианте (см. рис. 14.3).
- text_2 — это вторая строка текста в поисковом варианте (см. рис. 14.3).
- icon_1 — это пиктограмма, расположенная слева от варианта; обычно представляет собой ID ресурса.

- icon_2 — это пиктограмма, размещенная справа от варианта; обычно представляет собой ID ресурса.
- intent_action — это значение передается в случае активации действия через намерение. Это значение заменяет собой соответствующее действие из файла метаданных, ассоциированное с намерением (см. листинг 14.21).
- intent_data — это значение передается явлению SearchActivity, если намерение передает определенные данные. Это значение заменяет собой соответствующее действие из файла метаданных, ассоциированное с намерением (см. листинг 14.21). Это URI данных.
- intent_data_id — это значение прикрепляется к URI данных. Оно особенно полезно в тех случаях, когда требуется указать корневую часть информации в метаданных для одного раза, а потом менять эту информацию для каждого нового варианта. Такой способ позволяет оптимизировать работу.
- query — строка запроса, отправляемая поисковому явлению.
- shortcut_id — выше говорилось о том, что поисковый механизм Android кэширует варианты, предлагаемые поставщиком поиска. Такие кэшированные запросы называются «быстрыми вариантами». Если эта колонка отсутствует, Android будет кэшировать запросы, но никогда не станет требовать обновления кэша. Если в нем будет содержаться значение, эквивалентное SUGGEST_NEVER_MAKE_SHORTCUT, Android не станет кэшировать данный вариант. Если значение окажется другим, этот ID будет передан как последний сегмент пути в URI быстрого варианта (см. пункт «Виды URI поставщика поиска»).
- spinner_while_refreshing — это булево значение сообщает Android, следует ли использовать счетчик в процессе обновления быстрых вариантов.

Кроме того, существует изменяющийся набор других колонок, предназначенных для реагирования на клавиши действий. К этим колонкам мы обратимся позже, когда будем обсуждать сами клавиши действий. Теперь посмотрим, как наш пользовательский поставщик поиска возвращает такие колонки.

Заполнение и возвращение списка колонок

Ни один пользовательский поставщик поиска не должен возвращать все эти колонки. При работе с нашим поставщиком поиска мы будем возвращать только подмножество колонок, беря за основу набор функций, описанный в подразделе «Планирование пользовательского поставщика поиска».

В листинге 14.22 представлен список колонок, которые мы вырезали из листинга 14.20.

Листинг 14.22. Определение колонок поискового курсора

```
private static final String[] COLUMNS = {
   "_id", // эта колонка является обязательной
   SearchManager.SUGGEST_COLUMN_TEXT_1,
   SearchManager.SUGGEST_COLUMN_TEXT_2,
   SearchManager.SUGGEST_COLUMN_INTENT_DATA,
   SearchManager.SUGGEST_COLUMN_INTENT_ACTION,
   SearchManager.SUGGEST_COLUMN_SHORTCUT_ID
   };
```

Эти колонки выбраны для выполнения следующих функций.

Пользователь вводит слово с подсказкой, например great.m, в поле для быстрого поиска, и поставщик поиска не будет реагировать, пока во вводе не попадется первый образец «.». При обнаружении такого фрагмента поставщик содержимого извлечет из него слово (в данном случае great) и вернет два поисковых варианта.

Первый вариант откроет страницу с сайта thefreewebdictionary.com, на которой объясняется это слово, а второй вариант будет отправлен на сайт Google для поиска по образцу define:great.

Чтобы выполнить эту задачу, поставщик поиска загружает колонку intent_action в качестве intent.action.view, а также ассоциированные с намерением данные, в которых содержится весь URI. Скорее всего, встретив последовательность символов http://, с которых начинается URI данных, Android запустит браузер.

В колонку text1 мы запишем search some-website with:, а в text2 — само слово (в данном случае great). Кроме того, чтобы упростить ситуацию, мы зададим для быстрых вариантов значение SUGGEST_NEVER_MAKE_SHORTCUT. Эта настройка деактивирует кэширование и препятствует активации вводимых URI.

На этом мы завершаем анализ исходного кода пользовательского поставщика поиска. Вы узнали об URI, поисковых курсорах и поисковых метаданных, специфичных для поставщика поиска. Кроме того, вы выяснили, как заполняются поисковые колонки.

Теперь рассмотрим реализацию поискового явления для пользовательского поставщика поиска.

Реализация поискового явления для пользовательского поставщика поиска

Как мы уже говорили, при реализации пользовательского поставщика поиска необходимо внедрить в систему два компонента: сам поставщик поиска и явление, через которое будет осуществляться отклик на вводимые варианты. Предыдущий подраздел был посвящен работе с пользовательским поставщиком поиска. Далее рассмотрим соответствующее поисковое явление.

Как и в предыдущем подразделе, для начала обсудим основные функции поискового явления. Затем будет показан исходный код, позволяющий в общей перспективе оценить, как выполняются эти функции.

Задачи поискового явления

Обсуждая простой поставщик поиска, мы остановились лишь на нескольких задачах поискового явления. Теперь рассмотрим оставшиеся аспекты.

В процессе поиска Android активирует поисковое явление для реагирования на поисковые действия одним из двух способов. Активация может происходить либо при нажатии поисковой пиктограммы, расположенной в поле быстрого поиска, либо при непосредственном щелчке на варианте.

После активации поисковое явление обязательно должно проверить, почему оно было активировано. Эта можно определить по тому, какое действие ассоциировано с намерением, активировавшим явление. Таким образом, явление

проверяет, что это за действие. Часто оказывается, что действие — это ACTION_SEARCH. Однако поставщик поиска может переопределить его, прямо указав другое действие либо в файле метаданных поиска, либо в одной из колонок поискового курсора. Тип действия может быть любым. В нашем случае мы будем использовать действие VIEW.

Как было указано выше, при обсуждении простого поставщика поиска, мы также можем задать для режима запуска поискового явления значение singleTop. В таком случае поисковое явление будет выполнять дополнительную функцию — отвечать не только на onCreate(), но и на onNewIntent(). Мы рассмотрим оба этих варианта и покажем, насколько они похожи.

Итак, мы применим и onNewIntent(), и onCreate(), чтобы поэкспериментировать и с ACTION_SEARCH, и с ACTION_VIEW. Работая с действием поиска, мы просто отобразим текст запроса пользователю. Что касается действия просмотра, то мы передадим управление ситуацией браузеру и завершим актуальное явление. Из-за этого у пользователя сложится впечатление, как будто браузер активируется при простом нажатии на вариант.

Теперь рассмотрим исходный код файла SearchActivity.java.

Исходный код явления SearchActivity в пользовательском поставщике поиска

Вы уже знаете, какие задачи выполняет поисковое явление и, в частности, какие из них задействуются в нашем примере. Поэтому теперь можем перейти к изучению исходного кода этого поискового явления (листинг 14.23).

Листинг 14.23. SearchActivity

```
// файл: SearchActivity.java
public class SearchActivity extends Activity
{
   private final static String tag ="SearchActivity";
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      Log.d(tag,"I am being created");
      setContentView(R.layout.layout_test_search_activity);

      // здесь получаем и обрабатываем поисковый запрос
      final Intent queryIntent = getIntent();

      // действие-запрос
      final String queryAction = queryIntent.getAction();
      Log.d(tag,"Create Intent action:"+queryAction);

      final String queryString =
         queryIntent.getStringExtra(SearchManager.QUERY);
      Log.d(tag,"Create Intent query:"+queryString);

      if (Intent.ACTION_SEARCH.equals(queryAction))
      {
```

```
            this.doSearchQuery(queryIntent);
        }
        else if (Intent.ACTION_VIEW.equals(queryAction))
        {
            this.doView(queryIntent);
        }
        else {
            Log.d(tag,"Create intent NOT from search");
        }
        return;
    }

    @Override
    public void onNewIntent(final Intent newIntent)
    {
        super.onNewIntent(newIntent);
        Log.d(tag,"new intent calling me");

        // здесь получаем и обрабатываем поисковый запрос
        final Intent queryIntent = newIntent;

        // действие-запрос
        final String queryAction = queryIntent.getAction();
        Log.d(tag,"New Intent action:"+queryAction);

        final String queryString =
            queryIntent.getStringExtra(SearchManager.QUERY);
        Log.d(tag,"New Intent query:"+queryString);

        if (Intent.ACTION_SEARCH.equals(queryAction))
        {
            this.doSearchQuery(queryIntent);
        }
        else if (Intent.ACTION_VIEW.equals(queryAction))
        {
            this.doView(queryIntent);
        }
        else {
            Log.d(tag,"New intent NOT from search");
        }
        return;
    }
    private void doSearchQuery(final Intent queryIntent)
    {
        final String queryString =
            queryIntent.getStringExtra(SearchManager.QUERY);
        appendText("You are searching for:" + queryString);
    }
    private void appendText(String msg)
    {
        TextView tv = (TextView)this.findViewById(R.id.text1);
```

```
        tv.setText(tv.getText() + "\n" + msg);
    }
    private void doView(final Intent queryIntent)
    {
        Uri uri = queryIntent.getData();
        String action = queryIntent.getAction();
        Intent i = new Intent(action);
        i.setData(uri);
        startActivity(i);
        this.finish();
    }
}
```

Анализ исходного кода начнем с изучения того, как именно активируется это поисковое явление.

Подробности активации SearchActivity

Как и любое явление, которое встречалось нам ранее, поисковое явление должно активироваться намерением. Однако за активацию не всегда отвечает компонент намерения *action*. Оказывается, поисковое явление активируется явно, через его спецификацию имени компонента.

Может возникнуть вопрос: а почему это важно? Итак, мы знаем, что в одной из строк поставщика поиска мы специально указываем намерение action. Если это намерение VIEW, а данные, относящиеся к намерению, имеют вид HTTP URL, то непосвященный программист может подумать, что в ответ на это будет запускаться браузер, а не поисковое явление. Конечно, это было бы желательно. Но поскольку последнее намерение также загружается с именем компонента поискового явления, добавляемым к действию намерения и данным, имя компонента будет иметь приоритет.

Нам не совсем ясно, почему существует такое ограничение и можно ли его обойти. Но факт остается фактом — независимо от того, какое действие связывается с намерением в вашем поставщике поиска, активируется именно поисковое явление. По этой причине мы решили просто запускать браузер из поискового явления, а само поисковое явление — закрывать.

В качестве примера ниже приведено намерение, запускаемое Android для активации поиска после того, как мы нажимаем один из вариантов:

```
launching Intent {
act=android.intent.action.VIEW
dat=http://www.google.com
flg=0x10000000
cmp=com.ai.android.search.custom/.SearchActivity (has extras)
}
```

Обратите внимание на компонент spec, присутствующий в намерении. Он указывает прямо на поисковое явление. Поэтому независимо от того, какое именно действие вы укажете в связи с намерением, Android в любом случае запустит поисковое явление. В результате активация браузера становится одной из задач поискового явления.

Рассмотрим, что делать с этими намерениями в поисковом явлении.

Отклик на ACTION_SEARCH и ACTION_VIEW

Известно, что поисковое явление явно активируется по имени поисковой функцией Android. Однако вместе с активирующим намерением также передается указанное действие. Когда поле QSB активирует это явление через поисковую пиктограмму, мы имеем дело с действием ACTION_SEARCH.

Если это действие активируется одним из поисковых вариантов, оно может быть и другим. Это зависит от того, как поставщик поиска делает запрос. Ниже мы рассмотрим соответствующий фрагмент кода, чтобы разобраться, какой метод вызывать — метод запроса или метод просмотра (этот фрагмент присутствует в листинге 14.23).

```
if (Intent.ACTION_SEARCH.equals(queryAction))
{
   this.doSearchQuery(queryIntent);
}
else if (Intent.ACTION_VIEW.equals(queryAction))
{
   this.doView(queryIntent);
}
```

Из кода следует, что для просмотра активируется doView(), а для поиска — doSearchQuery().

В функции doView() мы получаем действие и URI данных и заполняем этими данными новое намерение, а затем активируем явление. Так запускается браузер. Мы завершаем явление, чтобы кнопка **Back** (Назад) снова возвращала нас к полю поиска, из которого мы активировали указанное явление.

В doSearchQuery() мы просто регистрируем текст поискового запроса для вида. Рассмотрим шаблон, используемый для поддержки doSearchQuery().

Шаблон поискового явления

В листинге 14.24 показан простой шаблон, используемый поисковым явлением в случае применения doSearchQuery(). Самый важный элемент выделен полужирным шрифтом.

Листинг 14.24. XML-шаблон явления SearchActivity

```
// файл: layout_search_activity.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:id="@+id/text1"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/search_activity_main_text"
   />
```

Здесь стоит также продемонстрировать файл strings.xml, обеспечивающий в этой программе выполнение некоторых текстовых функций.

Соответствующий файл strings.xml

Файл strings.xml, показанный в листинге 14.25, определяет текстовые строки для шаблона, а также такие компоненты, как имя приложения, несколько строк, конфигурирующих локальный поиск, и пр.

Листинг 14.25. Strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="search_activity_main_text">
   Это поисковое явление
   \n\n
   Этот код будет активироваться при использовании action_search
   а не action_view.
   \n\n
   action_search происходит при нажатии поисковой пиктограммы
   \n\n
   action_view происходит при нажатии на поисковый вариант
   </string>
   <string name="app_name">Custom Suggest Application</string>
   <string name="search_label">Custom Suggest Demo</string>
 <string name="search_hint">Custom Suggest Demo Hint</string>
</resources>
```

Отклик на onCreate() и onNewIntent()

Вновь обратимся к листингу 14.23. Важно отметить, что коды onCreate() и onNewIntent() почти идентичны. Такое случается нередко.

Если запускается поисковое явление, то, в зависимости от режима запуска этого явления, вызывается onCreate() или onNewIntent(). Если здесь не осуществить отклик, то можно пропустить активацию поиска.

ПРИМЕЧАНИЕ

В разделе «Ссылки» в конце этой главы приведен полезный источник по режимам запуска и onNewIntent().

Как завершать поисковое явление

Выше мы кратко упоминали о том, как делается отклик на doView(). В листинге 14.26 показан код для этой функции (извлеченный из листинга 14.23).

Листинг 14.26. Завершение поискового явления

```
private void doView(final Intent queryIntent)
{
   Uri uri = queryIntent.getData();
   String action = queryIntent.getAction();
   Intent i = new Intent(action);
   i.setData(uri);
   startActivity(i);
   this.finish();
}
```

Задача этой функции — запустить браузер. Если в конце не поставить `finish()`, то после нажатия кнопки **Back** (Назад) система вернет пользователя в поисковое явление, а не на предыдущий экран, чего ожидает пользователь.

В идеальном случае, чтобы сделать работу с программой максимально удобной, элемент управления никогда не должен подвергаться воздействию поискового явления. Чтобы этого не происходило, и применяется завершение явления. В предыдущем фрагменте кода вы также видите, как передать действие и данные намерения от оригинального намерения (заданного поставщиком поиска) к намерению браузера.

На этом мы завершаем обсуждение нескольких тем. Мы подробно рассмотрели реализацию поставщика поиска и реализацию поискового явления. Работая с этим материалом, мы также показали файл метаданных поиска и файл `strings.xml`. Далее мы завершим рассмотрение тех файлов, которые нужны нам для выполнения проекта из этой главы, и для этого обратимся к файлу описания, действующему на уровне приложения.

Файл описания пользовательского поставщика поиска

В файле описания объединяются многочисленные компоненты приложения. В примере с пользовательским поставщиком поиска, как и в других примерах, здесь объявляются различные составляющие, в частности поисковое явление и поставщик вариантов. Кроме того, в файле описания указывается, что эта программа предназначена для локального поиска — по умолчанию для осуществления поиска задается «поисковое явление».

В коде файла описания (листинг 14.27) эти фрагменты выделены полужирным шрифтом.

Листинг 14.27. Файл описания пользовательского поставщика поиска

```
// файл:manifest.xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.ai.android.search.custom"
      android:versionCode="1"
      android:versionName="1.0.0">
   <application android:icon="@drawable/icon"
              android:label="Custom Suggestions Provider">
<!--
****************************************************************
* Код, относящийся к поиску: поисковое явление.
****************************************************************
-->
   <activity android:name=".SearchActivity"
            android:label="Search Activity Label"
            android:launchMode="singleTop">
      <intent-filter>
         <action android:name="android.intent.action.SEARCH" />
         <category android:name="android.intent.category.DEFAULT" />
```

```
        </intent-filter>

        <meta-data android:name="android.app.searchable"
                   android:resource="@xml/searchable" />
    </activity>

<!-- Объявление стандартной функции поиска. -->
    <meta-data android:name="android.app.default_searchable"
               android:value=".SearchActivity" />
<!-- Объявление поставщика вариантов. -->
    <provider android:name="SuggestUrlProvider"
         android:authorities=
               "com.ai.android.search.custom.suggesturlprovider"
/>
</application>
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

Как видите, здесь выделены три вещи:

- определение поискового явления вместе с соответствующим XML-файлом поисковых метаданных;
- определение поискового явления как стандартной функции поиска в этом приложении;
- определение поставщика вариантов и его источника.

Подготовив весь исходный код, мы опробуем приложение и посмотрим, как оно выглядит в эмуляторе.

Опыт работы с пользовательским поставщиком поиска

Когда вы напишете и развернете эту программу при помощи ADT, никакие явления всплывать не будут, так как нет явления, которое можно было бы запустить. Вместо этого вы увидите, что программа успешно установлена в консоли Eclipse.

Это означает, что поставщик поиска готов отвечать на запросы, вводимые в глобальное поле QSB. Но прежде, чем это произойдет, нужно включить этот поставщик вариантов — чтобы он мог участвовать в глобальном поиске.

Выше в этой главе было показано, как попасть в раздел поисковых настроек приложения. Далее приведена быстрая комбинация, использующая ту самую поисковую функцию, которую мы уже изучили.

Откройте окно для глобального поиска и введите в него sett (рис. 14.30). В результате среди вариантов будет выдано приложение с настройками, и вы сможете его активировать.

Обратите внимание, как в данном случае для перехода в настройки используются изученные выше функции поля быстрого поиска. Придерживайтесь метода, описанного в начале этой главы, чтобы настроить это приложение для работы с вариантами. Когда все будет готово, введите текст в поле быстрого поиска (рис. 14.31).



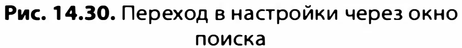


Рис. 14.30. Переход в настройки через окно поиска

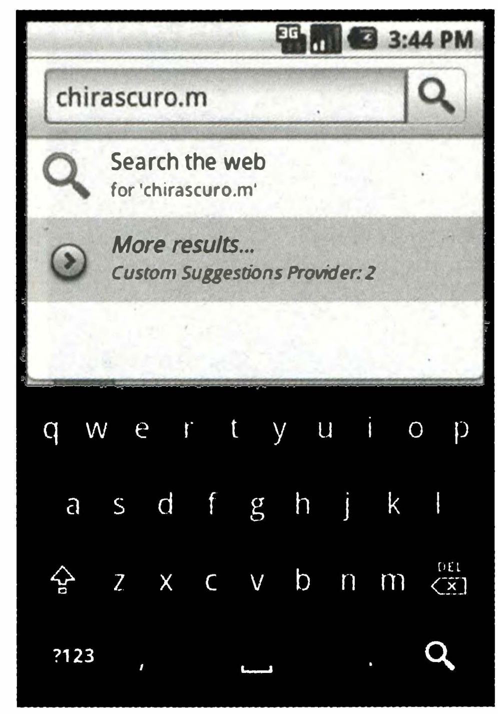


Рис. 14.31. Дополнительные результаты от пользовательского поставщика поиска

Обратите внимание, как пользовательский поставщик поиска демонстрирует варианты. Теперь, если перейти к одному из выданных поставщиком поиска вариантов и нажать пиктограмму быстрого поиска, Android переведет вас прямо в поисковое явление, не активируя никакого браузера (рис. 14.32).

В этом примере демонстрируется сравнение `ACTION_SEARCH` и `ACTION_VIEW`.

Когда вы воспользуетесь поставщиком поиска несколько раз, Android будет показывать варианты вместе с основным поиском, а не под кнопкой more (еще). На рис. 14.33 показан пример, в котором мы вводим `chiaroscuro.m` в поле для быстрого глобального поиска.

Обратите внимание на то, что поставщик вариантов выводится непосредственно, без подсказки more (еще). Теперь, нажав любую ссылку из freedictionary, вы попадаете в браузер (рис. 14.34).

Если выбрать вариант Google, вы попадете в окно браузера, показанное на рис. 14.35.

А на рис. 14.36 изображено, что случается, если не написать суффикс `.m`.

Обратите внимание: поставщик поиска не выдал никаких результатов.

На этом мы завершаем разговор о создании рабочего пользовательского поставщика поиска с нуля. Но, хотя мы и рассмотрели некоторые аспекты поиска, остается еще несколько вопросов, которые необходимо обсудить. Это клавиши действия (action keys) и специфичные для приложения (application-specific) поисковые данные. Они рассматриваются в следующем разделе.

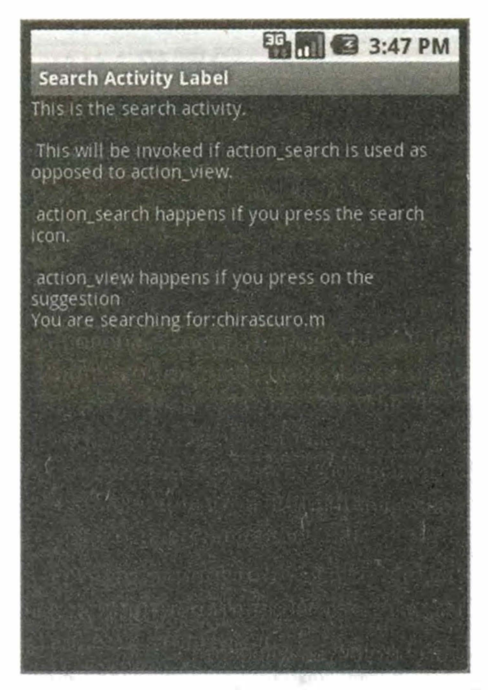


Рис. 14.32. Поисковый запрос, активирующий результаты поиска

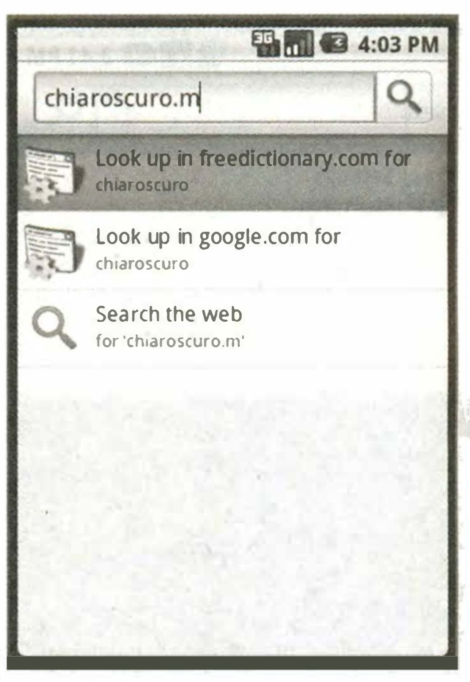


Рис. 14.33. Последовательность вариантов в поставщике поиска

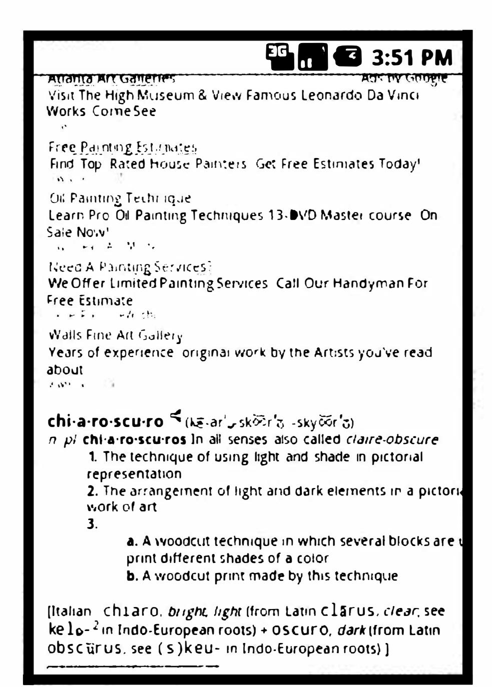


Рис. 14.34. Free dictionary



Рис. 14.35. Поиск определения в Google



Рис. 14.36. Пользовательский поставщик поиска без подсказки

Использование клавиш действия и специфичных для приложения поисковых данных

Клавиши действия и специфичные для приложения поисковые данные также повышают гибкость поиска в Android.

Клавиши действия позволяют использовать специальные клавиши действия при выполнении поисковых функций. Специфичные для программы поисковые данные дают возможность любому явлению передавать в поисковое явление дополнительную информацию.

Начнем с клавиш действия.

Применение клавиш действия при поиске в Android

Итак, вы уже знаете несколько способов активации поиска:

- при помощи пиктограммы, расположенной в поле для быстрого поиска;
- посредством поисковой клавиши, входящей в набор клавиш действия (на рис. 14.1 они показаны справа);
- используя специальную пиктограмму или кнопку, которая отображается в явлении;
- путем нажатия любой клавиши, для которой при объявлении Type-to-Search была специально задана такая функция.

В этом разделе мы рассмотрим другой способ активации поиска при помощи клавиш действия. Клавиши действия — это доступный на устройстве набор клавиш, выделенный для выполнения специальных функций. Некоторые из клавиш действия перечислены в листинге 14.28.

Листинг 14.28. Список кодов клавиш действия

```
keycode_dpad_up
keycode_dpad_down
keycode_dpad_left
keycode_dpad_right
keycode_dpad_center
keycode_back
keycode_call
keycode_camera
keycode_clear
kecode_endcall
keycode_home
keycode_menu
keycode_mute
keycode_power
keycode_search
keycode_volume_up
keycode_volume_down
```

Эти клавиши действия определяются в API для KeyEvent, доступном по следующей ссылке: http://developer.android.com/reference/android/view/KeyEvent.html.

ПРИМЕЧАНИЕ

Не все эти клавиши действия можно приспособить для поиска, но некоторые (например, keycode_call) для этого подходят. Попробуйте все и выберите те, которые могут быть вам полезны.

Определив, какие клавиши действия вы собираетесь использовать, можете сообщить Android, что вам нужна та или иная клавиша. Для этого ее код нужно поместить в метаданные, воспользовавшись XML-фрагментом, приведенным в листинге 14.29.

Листинг 14.29. Пример определения клавиши действия

```
<searchable xmlns:android="http://schemas.android.com/apk/res/android"
   android:label="@string/search_label"
   android:hint="@string/search_hint"
   android:searchMode="showSearchLabelAsBadge"

   android:includeInGlobalSearch="true"

   android:searchSuggestAuthority=
         "com.ai.android.search.simplesp.SimpleSuggestionProvider"
   android:searchSuggestSelection=" ? "
>
   <actionkey
      android:keycode="KEYCODE_CALL"
      android:queryActionMsg="call"
```

```
        android:suggestActionMsg="call"
        android:suggestActionMsgColumn="call_column" />

    <actionkey
        android:keycode="KEYCODE_DPAD_CENTER"
        android:queryActionMsg="doquery"
        android:suggestActionMsg="dosuggest"
        android:suggestActionMsgColumn="my_column" />
    .....
</searchable>
```

Кроме того, для одного и того же поискового контекста можно задать несколько клавиш действия. Ниже описано, за что отвечает каждый атрибут элемента actionKey и как он используется при отклике на нажатие клавиши.

- *Keycode* — это код клавиши в том виде, в котором он определен в классе API KeyEvent, который следует использовать для активации поискового явления. Существует два случая, в которых может быть нажата клавиша, идентифицируемая при помощи данного кода. Первый случай — когда пользователь вводит текст запроса в поле для быстрого поиска, но не переходит ни к одному из выбранных вариантов. Обычно если пользователю не предоставить на экране клавишу действия, он нажмет пиктограмму быстрого поиска. Если клавиша действия будет указана в поисковых метаданных Android, то пользователь сможет нажать не только «лупу», но и эту клавишу. Второй случай — когда пользователь выбирает один из вариантов, предложенных поставщиком поиска, а затем нажимает клавишу действия. В обоих случаях поисковое явление активируется действием ACTION_SEARCH. Чтобы мы знали, что это действие активировано при помощи специальной клавиши, мы должны найти дополнительную строку — SearchManager.ACTION_KEY. Если здесь есть значение, то мы знаем, что вызов был сделан в ответ на нажатие клавиши.
- queryActionMsg — любой текст, указываемый в этом элементе, передается намерению, которое активирует поисковое явление, в виде дополнительной строки, называемой SearchManager.ACTION_MSG. Если вы получаете это сообщение от намерения и оно не отличается от того, что вы указали в метаданных, то вы знаете, что вызов пришел прямо из QSB в результате нажатия клавиши действия. Без такой проверки нельзя узнать, было ли действие ACTION_SEARCH вызвано нажатием клавиши или предложенного варианта.
- suggestActionMsg — любой текст, вводимый в этот элемент, передается намерению, которое активирует поисковое явление, в виде дополнительной строки, называемой SearchManager.ACTION_MSG. Дополнительные строки для этого случая и для queryActionMsg одинаковы. Если в обоих полях указать одинаковое значение, например call, то вы не узнаете, каким именно способом пользователь активировал клавишу действия. Во многих случаях это не имеет значения, поэтому можно просто задать в обоих случаях одно и то же значение. Но если различение необходимо, то вы должны обозначить его отдельно.
- suggestActionMsgColumn — значения queryActionMsg и suggestActionMsg глобально применяются и к поисковому явлению, и к поставщику поиска. Не существует

способа изменить значение действия в зависимости от конкретного варианта, данного поставщиком. Если вам потребуется совершить такую операцию, то нужно будет задать в метаданных дополнительную колонку для курсора. Таким образом, Android сможет взять текст из этой дополнительной колонки и передать его явлению в рамках активации намерения ACTION_SEARCH. Интересно отметить, что значение этой дополнительной колонки передается в той же дополнительной строке, находящейся в намерении, а именно — SearchManager.ACTION_MSG.

Из этих атрибутов обязательным является код клавиши. Кроме того, для работы клавиши действия необходимо, чтобы присутствовал еще хотя бы один из трех остальных атрибутов.

Если вы собираетесь работать с suggestActionMsgColumn, то нужно будет заполнить эту колонку в классе поставщика поиска. Если вы собираетесь использовать обе эти клавиши, то в листинге 14.29 нужно добавить две дополнительные колонки для строковых значений, определяемые в курсоре вариантов (см. листинг 14.22), а именно — call_column и my_column. В таком случае массив колонок курсора будет выглядеть, как в листинге 14.30.

Листинг 14.30. Пример указания колонок для клавиш действия в курсоре для вариантов

```
private static final String[] COLUMNS = {
     "_id", // должен содержать эту колонку
     SearchManager.SUGGEST_COLUMN_TEXT_1,
     SearchManager.SUGGEST_COLUMN_TEXT_2,
     SearchManager.SUGGEST_COLUMN_INTENT_DATA,
     SearchManager.SUGGEST_COLUMN_INTENT_ACTION,
     SearchManager.SUGGEST_COLUMN_SHORTCUT_ID,
 "call_column",
 "my_column"
     };
```

Работа с контекстом поиска, специфичным для конкретного приложения

Поисковый механизм Android позволяет явлению передавать поисковому явлению при его активации дополнительные данные, связанные с поиском. Далее мы подробно рассмотрим этот процесс.

Как уже было указано, явление нашей программы может переопределить метод onSearchRequested(), чтобы поиск деактивировался в случае возвращения значения false. Интересно отметить, что тот же метод также можно использовать для передачи поисковому явлению дополнительных данных, специфичных для конкретного приложения. В листинге 14.31 показан соответствующий пример.

Листинг 14.31. Передача дополнительного контекста

```
public boolean onSearchRequested()
{
   Bundle applicationData = new Bundle();
   applicationData.putString("string_key","some string value");
```

```
    applicationData.putLong("long_key",290904);
    applicationData.putFloat("float_key",2.0f);

    startSearch(null,     // первичная строка поискового запроса
    false,   // не выбирать "select initial query"
    applicationData,     // дополнительные данные
    false // не инициировать глобальный поиск принудительно
    );

    return true;
}
```

ПРИМЕЧАНИЕ

Можете воспользоваться справкой по следующему API Bundle, в которой описаны различные функции, применимые к объекту «пакет» (bundle): http://developer.android.com/reference/android/os/Bundle.html.

Когда поиск запущен таким способом, явление может использовать вызванный дополнительно `SearchManager.APP_DATA`, чтобы получить из приложения пакет данных. В листинге 14.32 показано, как можно получить данные из каждого указанного выше поля.

Листинг 14.32. Получение дополнительного контекста

```
Bundle applicationData =
    queryIntent.getBundleExtra(SearchManager.APP_DATA);
if (applicationData != null)
{
    String s = applicationData.getString("string_key");
    long   l = applicationData.getLong("long_key");
    float f = applicationData.getFloat("float_key");
}
```

Рассмотрим метод `startSearch()`. Он описан в рамках API Activity по следующей ссылке: **http://developer.android.com/reference/android/app/Activity.html.**

Он может принимать четыре следующих аргумента:

- `initialQuery` // строковый аргумент;
- `selectInitialQuery` // булев аргумент;
- `applicationDataBundle` // пакет с данными;
- `globalSearchOnly` // булев аргумент.

Первый аргумент, если он присутствует, заносит в поле для быстрого поиска текст запроса.

Если второй булев аргумент имеет значение `true`, он подсветит текст. Таким образом, пользователь сможет заменить весь выделенный текст запроса новым. При значении `false` курсор окажется в самом конце текста запроса.

Третий аргумент — это пакет с данными, который мы готовим.

Четвертый аргумент, имея значение `true`, всегда будет активировать глобальный поиск. При значении `false` сначала будет активирован локальный поиск — если он доступен. Если нет, то активируется глобальный поиск.

Ресурсы

Завершая эту главу, мы хотели бы порекомендовать вам ресурсы, которые помогли нам написать ее.

По следующей ссылке находится основная документация от Google, посвященная поиску в Android. Этот же сайт может послужить справкой по API для основной поисковой функции Android, то есть для SearchManager: http://developer.android.com/reference/android/app/SearchManager.html.

При создании поисковых явлений бывает полезно задавать их как singleTop, в результате чего генерируется onNewIntent(). Подробнее об этом методе рассказано по адресу http://developer.android.com/reference/android/app/Activity.html#onNewIntent(android.content.Intent).

По следующей ссылке размещен пример реализации поставщика поиска авторства специалистов Google. Ссылка указывает прямо на исходный код этой реализации: http://developer.android.com/guide/samples/SearchableDictionary/index.html.

Об API для поиска недавно вводившихся вариантов (Search Recent Suggestions) написано здесь: http://developer.android.com/reference/android/provider/SearchRecentSuggestions.html.

Материал, размещенный по следующему адресу, поможет лучше понять явления, задачи и режимы запуска, в частности режим запуска singleTop, часто используемый в поисковых явлениях: http://developer.android.com/guide/topics/fundamentals.html.

По следующей ссылке дается справка об API Bundle, в частности, рассмотрены различные функции, используемые с объектом Bundle. Он полезен при работе с поисковыми данными, специфичными для конкретных приложений: http://developer.android.com/reference/android/os/Bundle.html.

Далее приведен адрес, по которому авторы разместили свои исследования, касающиеся поиска в Android. Мы будем обновлять материалы и после того, как выйдет эта книга. Кроме того, здесь есть дополнительные ссылки, указывающие, где можно скачать проекты, выполненные в этой главе. http://www.satyakomatineni.com/akc/display?url=NotesIMPTitlesURL&ownerUserId=satya&folderName=Android%20Search.

Резюме

В этой главе мы детально рассмотрели, как внутри системы Android осуществляются функции поиска. Вы узнали, как явления и поставщики поиска взаимодействуют с поиском Android. Было показано, как использовать SearchRecentSuggestionsProvider.

Мы с нуля написали пользовательский поставщик поиска, в процессе продемонстрировали поисковый курсор и подробно рассмотрели его колонки. Мы исследовали URI, отвечающие за получение данных от поставщиков поиска. Было приведено много примеров кода, которые позволят вам без труда разработать и применить ваши собственные поисковые стратегии.

Пользуясь гибкостью одного только поискового курсора, Android превращает обычный поиск в настоящий рог изобилия, из которого сыплется информация и который всегда у вас под рукой.

15 Исследование текста для работы с API синтеза речи и интерфейсами машинного перевода

Версии Android выше 1.6 используют многоязычный синтезатор речи, называемый Pico. Благодаря ему любая программа Android способна произносить текстовые строки с акцентом, соответствующим выбранному языку. Механизм синтеза речи позволяет пользователям работать с устройством, не глядя на экран. На мобильной платформе такая функция может быть исключительно важной. Вдумайтесь, сколько человек попали под машину, пока читали SMS? Не лучше ли, если SMS можно будет просто слушать? Кроме того, можно слушать экскурсионный материал, осматривая достопримечательности, а не читать его. Можно представить себе очень много случаев, в которых программа станет гораздо функциональнее, если обеспечить в ней голосовые функции. В этой главе будет изучен класс Android `TextToSpeech`. Вы узнаете, что нужно сделать, чтобы гаджет зачитывал текст, написанный на экране.

В этой главе также будет описано взаимодействие с онлайн-API Google, предназначенным для перевода текста с одного языка на другой. Эта возможность стала доступна совсем недавно.

Основы синтеза речи в Android

Прежде чем мы перейдем к интегрированию в программу функции преобразования текста в речь (Text to Speech), послушаем, как она работает. На эмуляторе или на устройстве (Android 1.6 и выше) перейдите к основному экрану настроек (Settings) и выберите Text to Speech (Преобразование текста в речь). В некоторых версиях Android эта функция может называться Speech synthesis. Там есть команда Listen to an example (Прослушать образец). Щелкните на ней — и услышите слова: This is an example of speech synthesis in English (Это пример синтеза речи на английском языке). Обратите внимание и на другие присутствующие здесь команды (рис. 15.1).

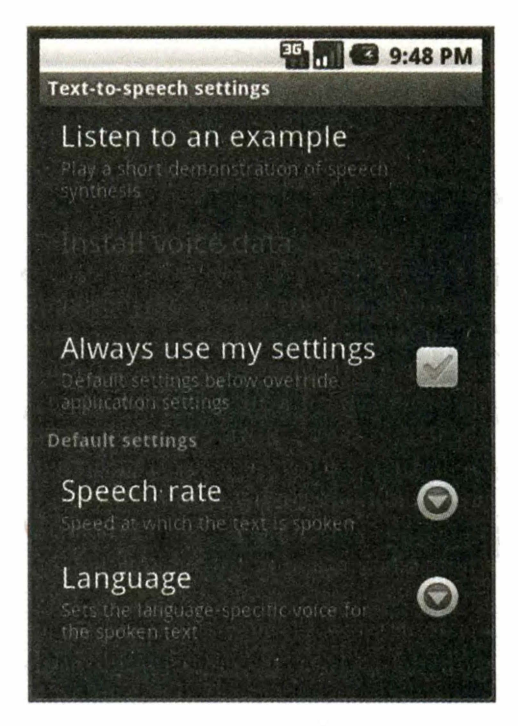


Рис. 15.1. Экран настроек для преобразования текста в речь

Здесь можно менять язык и темп речи. Языковая команда изменяет не только произносимые слова, но и акцент, хотя перевод остается прежним — «Это пример синтеза речи» на выбранный вами язык. Прочитанный текст переводится, и акцент также изменяется в соответствии с параметром, указанным в разделе Language (Язык). Учитывайте, что функция синтеза речи отвечает только за голосовую часть задачи. Перевод осуществляется службой Google Translate, которую мы рассмотрим во второй части этой главы. Далее, когда мы перейдем к реализации функции синтеза речи в нашей программе, нам потребуется синхронизировать голос с языком — так, чтобы, например, французский текст произносился по-французски. Темп речи варьируется от Very slow (Очень медленно) до Very fast (Очень быстро). Осторожно обращайтесь с параметром Always use my settings (Всегда использовать мои настройки). Если вы или пользователь установите этот флажок, то, возможно, программа начнет работать не так, как вы ожидаете, поскольку эти настройки могут получить приоритет над теми, которые вы хотите применить в своей программе.

Попробуем разобраться, что происходит, когда мы экспериментируем с этими настройками синтеза речи. «За кулисами» Android запускает Pico, многоязычный движок для синтеза речи. Явление с настройками, в котором мы находимся, инициализирует эту службу для выбранного нами языка и темпа речи. Когда мы нажимаем Listen to an example (Прослушать пример), явление с настройками отправляет текст движку, а система произносит его через аудиовыход. Pico разбивает текст на мелкие фрагменты, которые умеет произносить, и объединяет их таким образом, что речь звучит очень натурально. Логика, по которой работает эта система, конеч-

но же, гораздо сложнее, чем мы описали, но нам на данном этапе можно просто считать, что все происходит по волшебству. К счастью, это волшебство почти не занимает дискового пространства и памяти, поэтому Pico — идеальное дополнение к мобильному телефону.

В каждом устройстве может быть установлен только один движок для синтеза речи. Все явления устройства совместно используют этот механизм, необходимо об этом помнить. Вместе с тем это означает, что мы не можем точно знать, когда наш текст будет произнесен и произойдет ли это вообще. Но в интерфейсе системы синтеза речи предусмотрены обратные вызовы, поэтому мы можем иметь представление о том, что происходит с текстом, который мы собираемся воспроизвести.

В данном примере мы собираемся создать приложение, которое будет зачитывать нам введенный нами текст. Оно очень простое, и мы пишем его, чтобы показать, как просто настраивается преобразование текста в речь. Для начала создадим новый проект Android, пользуясь артефактами, приведенными в листинге 15.1.

Листинг 15.1. Код XML и Java для демонстрационного приложения Simple TTS

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

   <EditText android:id="@+id/wordsToSpeak"
      android:hint="Type words to speak here"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"/>

   <Button android:id="@+id/speak"
      android:text="Speak"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:enabled="false" />

</LinearLayout>

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.speech.tts.TextToSpeech.OnInitListener;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends Activity implements OnInitListener {
   private EditText words = null;
```

```
private Button speakBtn = null;
private static final int REQ_TTS_STATUS_CHECK = 0;
private static final String TAG = "TTS Demo";
private TextToSpeech mTts;

/** Вызывается при первом создании явления. */
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   words = (EditText)findViewById(R.id.wordsToSpeak);
   speakBtn = (Button)findViewById(R.id.speak);
   speakBtn.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View view) {
         mTts.speak(words.getText().toString(),
               TextToSpeech.QUEUE_ADD, null);
      }});

   // Проверяем, чтобы удостовериться, что система синтеза речи
   // присутствует и готова к использованию.
   Intent checkIntent = new Intent();
   checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
   startActivityForResult(checkIntent, REQ_TTS_STATUS_CHECK);
}

protected void onActivityResult(int requestCode,
         int resultCode, Intent data) {
   if (requestCode == REQ_TTS_STATUS_CHECK) {
      switch (resultCode) {
      case TextToSpeech.Engine.CHECK_VOICE_DATA_PASS:
         // Система синтеза речи настроена и работает.
         mTts = new TextToSpeech(this, this);
         Log.v(TAG, "Pico is installed okay");
         break;
      case TextToSpeech.Engine.CHECK_VOICE_DATA_BAD_DATA:
      case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_DATA:
      case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_VOLUME:
         // недостающие данные, установите
         Log.v(TAG, "Need language stuff: " + resultCode);
         Intent installIntent = new Intent();
         installIntent.setAction(
            TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
         startActivity(installIntent);
         break;
      case TextToSpeech.Engine.CHECK_VOICE_DATA_FAIL:
      default:
         Log.e(TAG, "Got a failure. TTS apparently not available");
      }
   }
```

```
        else {
        // в другом случае
        }
    }

    @Override
    public void onInit(int status) {
        // Теперь, когда система синтеза речи готова, активируем кнопку.
        if( status == TextToSpeech.SUCCESS) {
            speakBtn.setEnabled(true);
        }
    }

    @Override
    public void onPause()
    {
        super.onPause();
        // Если теряем фокус, прекращаем говорить.
        if( mTts != null)
            mTts.stop();
    }

    @Override
    public void onDestroy()
    {
        super.onDestroy();
        mTts.shutdown();
    }
}
```

В качестве пользовательского интерфейса здесь применяется обычный вид EditText, где мы можем набирать слова, которые затем будут произноситься, а также кнопка, активирующая речь (рис. 15.2). С кнопкой ассоциирован метод onClick(), забирающий текстовую строку из вида EditText и ставящий ее в очередь на обработку к системе синтеза речи — это делается при помощи метода speak() со значением QUEUE_ADD. Не забывайте, что механизм синтеза речи используется совместно несколькими явлениями, поэтому здесь и применяется очередь, в которую ставится текст, подготовленный для воспроизведения, — чтобы дождаться завершения выполнения текущей задачи (правда, такая ситуация очередности обычно не возникает). Другой параметр — QUEUE_FLUSH, позволяющий сбросить из очереди весь имеющийся текст и немедленно воспроизвести конкретный фрагмент. По завершении метода onCreate() мы инициируем намерение, отправляющее запрос к движку синтеза речи. Так мы узнаем, все ли в порядке с текстом, который предстоит воспроизвести. Поскольку мы хотим получить ответ, используем метод startActivityForResult(), в котором передаем код запроса. Ответ мы получаем в методе onActivityResult(), где нужно найти CHECK_VOICE_DATA_PASS. Поскольку в механизме синтеза речи предусмотрено несколько типов resultCode, означающих «OK», мы не можем искать просто RESULT_OK. Далее перечислены другие значения, которые могут встретиться в операторе-переключателе.

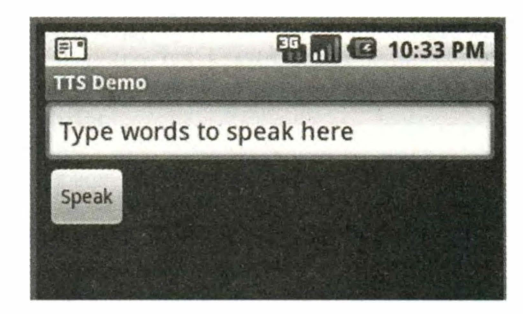


Рис. 15.2. Пользовательский интерфейс демонстрационного механизма синтеза речи

Если в ответ мы получим CHECK_VOICE_DATA_PASS, то следует инстанцировать объект TextToSpeech. Обратите внимание: в явлении MainActivity используется OnInitListener. Таким образом, мы получаем обратный вызов после того, как будет создан и станет доступным интерфейс механизма синтеза речи. Для этого применяется метод onInit(). Если в onInit() придет значение SUCCESS, это значит, что система готова произносить текст, и мы активируем кнопку, находящуюся в пользовательском интерфейсе. Следует отметить вызов stop() в onPause() и shutdown() в onDestroy(). Если мы вызываем stop(), у программы появляется более приоритетная задача, механизм синтеза речи выходит из фокуса и речь должна прерваться. Но мы не хотим прерывать какой-нибудь другой аудиосигнал, выполняемый в другом явлении, которое стало приоритетным. Метод shutdown() вызывается для того, чтобы уведомить Android, что работа движка TTS приостановлена и ресурсы можно высвободить.

Поэкспериментируем с этим примером, разберем различные предложения и фразы. Сейчас попробуем прослушать большой фрагмент текста несколько раз. Затем представим, что произойдет, если работа программы будет прервана в ходе чтения большого фрагмента текста, например в том случае, когда другое приложение обратилось к TTS со значением QUEUE_FLUSH, или просто при потере фокуса. Чтобы смоделировать такую ситуацию, нажмите кнопку Home (Домой) во время чтения большого блока текста. Посокольку мы вызываем stop() в onPause(), речь прекращается, а наша программа при этом продолжает работать в фоновом режиме. Если механизм синтеза речи вернется в фокус, то как мы узнаем, где остановились? Было бы неплохо предусмотреть способ узнать, где остановилась речь, или как минимум оказаться рядом с этим местом. Такой способ есть, но над его реализацией потребуется немного поработать.

Использование фрагментов речи для отслеживания речевой информации

Движок TTS может запускать в программе обратный вызов, когда завершится произнесение небольшого текста, который в контексте TTS называется *речевым фрагментом* (utterance). Обратный вызов осуществляется при помощи метода setOnUtteranceCompletedListener(), применяемого к экземпляру TTS; в нашем при-

мере — mTts. При вызове speak() можно добавить пару «имя — значение», приказывающую движку TTS дать нам знать, что воспроизведение речевого фрагмента завершится. Устанавливая для движка TTS уникальные ID речевых фрагментов, мы можем отслеживать, какие фрагменты уже произнесены, а какие — нет. Если программа вернется в фокус после того, как воспроизведение речи было прервано, мы возобновим текст с того фрагмента речи, который следует за последним произнесенным до конца. Основываясь на предыдущем примере, изменим код в соответствии с листингом 15.2.

Листинг 15.2. Изменения, вносимые в MainActivity и иллюстрирующие отслеживание речевых фрагментов

```
import java.util.HashMap;
import java.util.StringTokenizer;

public class MainActivity extends Activity implements OnInitListener, ➥
OnUtteranceCompletedListener

   private int uttCount = 0;
   private int lastUtterance = -1;
   private HashMap<String, String> params = new HashMap<String, String>();

      @Override
      public void onClick(View view) {
         StringTokenizer st = new
               StringTokenizer(words.getText().toString(),",.");
         while (st.hasMoreTokens()) {
            params.put(TextToSpeech.Engine.KEY_PARAM_UTTERANCE_ID,
                     String.valueOf(uttCount++));
            mTts.speak(st.nextToken(), TextToSpeech.QUEUE_ADD, params);;
         }

         @Override
         public void onInit(int status) {
            // Теперь, когда система синтеза речи готова,
            // активируем кнопку.
            if( status == TextToSpeech.SUCCESS) {
               speakBtn.setEnabled(true);
               mTts.setOnUtteranceCompletedListener(this);
            }
         }

         @Override
         public void onUtteranceCompleted(String uttId) {
            Log.v(TAG, "Got completed message for uttId: " + uttId);
            lastUtterance = Integer.parseInt(uttId);
         }
```

В данном случае первым делом нужно убедиться, что в нашем явлении MainActivity также реализуется интерфейс OnUtteranceCompletedListener. При этом мы сможем получать от TTS обратные вызовы тогда, когда будет завершаться произнесение

определенного речевого фрагмента. Кроме того, нам потребуется модифицировать наш метод кнопки `onClick()`, чтобы сообщать в нем дополнительную информацию, при помощи которой ID будет ассоциироваться с каждым отправляемым текстовым фрагментом. В этой новой версии нашего примера мы разобьем текст на фрагменты, применив точки и запятые в качестве разделительных знаков. Затем обработаем в цикле все наши фрагменты через цикл. При этом метод будет иметь значение `QUEUE_ADD`, а не `QUEUE_FLUSH` (мы не хотим прерывать сами себя!). С каждым речевым фрагментом будет передаваться уникальный ID. Мы применим обычный инкрементный счетчик, но, разумеется, преобразуем данные в тип `String`. С каждым ID фрагмента речи может быть связан любой текст, так как данные являются строковыми; мы не ограничены одними только цифрами. На самом деле в качестве ID речевого фрагмента можно использовать даже саму строку, но строки, предназначенные для произнесения, обычно очень длинны, а такие большие ID будут снижать производительность работы. Потребуется изменить метод `onInit()`, чтобы зарегистрироваться для получения обратных вызовов от завершенных речевых фрагментов, и, наконец, следует предоставить метод обратного вызова `onUtteranceCompleted()` для движка TTS. Этот метод будет активироваться, когда завершится произнесение речевого фрагмента. В данном примере мы просто будем регистрировать в LogCat сообщение о завершении каждого фрагмента.

При запуске этого нового примера введите какой-нибудь текст, содержащий точки и запятые, а затем нажмите кнопку **Speak** (Речь). Когда будете слушать, как произносится текст, следите за окном LogCat. Вы увидите, что фрагменты текста сразу же выставляются в очередь, а когда завершается произнесение определенного фрагмента, активируется обратный вызов и для каждого законченного фрагмента речи регистрируется сообщение. Если прервать выполнение этого примера, например нажав кнопку **Home** (Домой), пока текст читается, остановятся и голос, и обратные вызовы. Теперь точно известно, какой фрагмент речи произносился последним, и позже, возобновив работу с программой, мы вернемся именно к нему.

Использование аудио при работе с голосом

В системе TTS предусмотрена функция исправления слов или фраз, которые при стандартных настройках будут произноситься неверно. Например, если написать `Don Quixote` в качестве текста для произнесения, то система прочитает это имя неправильно. TTS очень хорошо угадывает, как должны произноситься слова, но не может знать всех исключений из правил. Как же с этим справиться? Один из способов — записать аудиофрагмент, который будет воспроизводиться, если нужно произнести нестандартное слово. Чтобы голос при этом не отличался от стандартного, звук должна синтезировать уже TTS. Поэтому запишем фрагмент, а затем сообщим TTS, что в определенном случае он должен воспроизвести этот фрагмент, а не действовать по умолчанию. Весь фокус в том, чтобы текст звучал именно так, как нам нужно. Приступим.

В Eclipse создаем новый проект Android. Используем XML-файл из листинга 15.3, чтобы создать основной шаблон. В данном случае применим простой подход — скопируем текст прямо в файл шаблона, а не будем делать ссылки на строки.

Как правило, в файле шаблона используются ID строковых ресурсов. Шаблон будет выглядеть так, как на рис. 15.3.

Листинг 15.3. XML-шаблон для воспроизведения вместо текста записанного аудиофрагмента

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">

  <EditText android:id="@+id/wordsToSpeak"
    android:text="Dohn Keyhotay"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>

  <Button android:id="@+id/speakBtn"
    android:text="Speak"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:enabled="false" />

  <TextView android:id="@+id/filenameLabel"
    android:text="Filename:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

  <EditText android:id="@+id/filename"
    android:text="/sdcard/donquixote.wav"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>

  <Button android:id="@+id/recordBtn"
    android:text="Record"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:enabled="false" />

  <Button android:id="@+id/playBtn"
    android:text="Play"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:enabled="false" />

  <TextView android:id="@+id/useWithLabel"
    android:text="Use with:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

  <EditText android:id="@+id/realText"
    android:text="Don Quixote"
    android:layout_width="fill_parent"
```



Рис. 15.3. Пользовательский интерфейс демонстрационного TTS, ассоциирующий звуковой файл и текст

```
        android:layout_height="wrap_content"/>

    <Button android:id="@+id/assocBtn"
        android:text="Associate"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:enabled="false" />

</LinearLayout>
```

Нам нужно поле, в котором будет содержаться специальный текст, записываемый при помощи TTS в звуковой файл. В шаблоне также указывается имя этого файла. Наконец, нужно ассоциировать звуковой файл с той строкой, которая будет в нем воспроизводиться.

Теперь рассмотрим код Java для `MainActivity` (листинг 15.4). В методе `onCreate()` устанавливаются обработчики щелчков на кнопках **Speak** (Речь), **Play** (Воспроизведение), **Record** (Запись) и **Associate** (Ассоциирование), а затем при помощи намерения запускается движок TTS. Оставшийся код состоит из обратных вызовов, обрабатывающих результаты, которые получаются от намерения. Это намерение проверяет, правильно ли настроен TTS, а обратные вызовы обрабатывают результат инициализации движка TTS и обычные обратные вызовы. Обычные обратные вызовы предназначены для запуска и остановки работы нашего явления.

Листинг 15.4. Код Java для демонстрации аудио, сохраненного для текстового фрагмента

```
import java.io.File;
import android.app.Activity;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.speech.tts.TextToSpeech.OnInitListener;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends Activity implements OnInitListener {
   private EditText words = null;
   private Button speakBtn = null;
   private EditText filename = null;
   private Button recordBtn = null;
   private Button playBtn = null;
   private EditText useWith = null;
   private Button assocBtn = null;
   private String soundFilename = null;
   private File soundFile = null;
   private static final int REQ_TTS_STATUS_CHECK = 0;
   private static final String TAG = "TTS Demo";
   private TextToSpeech mTts = null;
   private MediaPlayer player = null;

   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      words = (EditText)findViewById(R.id.wordsToSpeak);
      filename = (EditText)findViewById(R.id.filename);
      useWith = (EditText)findViewById(R.id.realText);

      speakBtn = (Button)findViewById(R.id.speakBtn);
      speakBtn.setOnClickListener(new OnClickListener() {
         @Override
         public void onClick(View view) {
            mTts.speak(words.getText().toString(),
                  TextToSpeech.QUEUE_ADD, null);
         }});

      recordBtn = (Button)findViewById(R.id.recordBtn);
```

```
recordBtn.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View view) {
      soundFilename = filename.getText().toString();
      soundFile = new File(soundFilename);
      if (soundFile.exists())
         soundFile.delete();

      if(mTts.synthesizeToFile(words.getText().toString(),
               null, soundFilename)
            == TextToSpeech.SUCCESS) {
         Toast.makeText(getBaseContext(),
            "Sound file created",
            Toast.LENGTH_SHORT).show();
         playBtn.setEnabled(true);
         assocBtn.setEnabled(true);
      }
      else {
         Toast.makeText(getBaseContext(),
               "Oops! Sound file not created",
               Toast.LENGTH_SHORT).show();
      }
   }});

playBtn = (Button)findViewById(R.id.playBtn);
playBtn.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View view) {
   try {
      player = new MediaPlayer();
      player.setDataSource(soundFilename);
      player.prepare();
      player.start();
   }
   catch(Exception e) {
      Toast.makeText(getBaseContext(),
         "Hmmmmm. Can't play file",
         Toast.LENGTH_SHORT).show();
      e.printStackTrace();
   }
}});

assocBtn = (Button)findViewById(R.id.assocBtn);
assocBtn.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View view) {
      mTts.addSpeech(useWith.getText().toString(), soundFilename);
      Toast.makeText(getBaseContext(),
         "Associated!",
         Toast.LENGTH_SHORT).show();
   }});
```

```
        // Проверяем, что система синтеза речи присутствует
        // и готова к использованию.
        Intent checkIntent = new Intent();
        checkIntent.setAction(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
        startActivityForResult(checkIntent, REQ_TTS_STATUS_CHECK);
    }

    protected void onActivityResult(int requestCode, int resultCode,
                                Intent data) {
        if (requestCode == REQ_TTS_STATUS_CHECK) {
            switch (resultCode) {
            case TextToSpeech.Engine.CHECK_VOICE_DATA_PASS:
                // Система синтеза речи настроена и работает.
                mTts = new TextToSpeech(this, this);
                Log.v(TAG, "Pico is installed okay");
                break;
            case TextToSpeech.Engine.CHECK_VOICE_DATA_BAD_DATA:
            case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_DATA:
            case TextToSpeech.Engine.CHECK_VOICE_DATA_MISSING_VOLUME:
                // недостающие данные, установите
                Log.v(TAG, "Need language stuff: " + resultCode);
                Intent installIntent = new Intent();
                installIntent.setAction(
                    TextToSpeech.Engine.ACTION_INSTALL_TTS_DATA);
                startActivity(installIntent);
                break;
            case TextToSpeech.Engine.CHECK_VOICE_DATA_FAIL:
            default:
                Log.e(TAG, "Got a failure. TTS apparently not available");
            }
        }
        else {
            // в другом случае
        }
    }

    @Override
    public void onInit(int status) {
        // Теперь, когда система синтеза речи готова, активируем кнопки.
        if( status == TextToSpeech.SUCCESS) {
            speakBtn.setEnabled(true);
            recordBtn.setEnabled(true);
        }
    }

    @Override
    public void onPause()
    {
        super.onPause();
        // Если теряем фокус, прекращаем воспроизведение.
```

```
        if(player != null) {
            player.stop();
        }
        // Если теряем фокус, прекращаем говорить.
        if( mTts != null)
            mTts.stop();
        }

    @Override
    public void onDestroy()
    {
        super.onDestroy();
        if(player != null) {
            player.release();
        }
        if( mTts != null) {
            mTts.shutdown();
        }
    }
}
```

Чтобы этот пример работал, в файл описания Android.xml требуется добавить специальное право доступа — android.permission.WRITE_EXTERNAL_STORAGE. При запуске этого примера вы должны видеть пользовательский интерфейс, как на рис. 15.3.

Мы собираемся записать текст, который при воспроизведении будет звучать как Don Quixote, но реальных слов мы использовать не можем. Нужно составить текст, который будет звучать так, как нам требуется. Чтобы прослушать, как звучат такие искусственные слова, нажмите кнопку **Speak** (Речь). Очень даже ничего! Затем нажмите **Record** (Запись), чтобы записать аудио в файл WAV. Если запись пройдет успешно, активируются кнопки **Play** (Воспроизведение) и **Associate** (Ассоциирование). Чтобы прослушать файл WAV непосредственно при помощи MediaPlayer, нажмите кнопку **Play** (Воспроизведение). Если вам нравится то, что получилось, нажмите кнопку **Associate** (Ассоциирование). Таким образом, активируется метод addSpeech() системы TTS, который прикрепляет новый звуковой файл к строке в поле **Use with** (Использовать с). Если этот процесс завершится успешно, вернитесь к верхнему виду EditText, введите в нем Don Quixote и нажмите **Speak** (Речь). Теперь имя звучит так, как нужно. Обратите внимание, что метод synthesizeToFile() сохраняет файлы только в формате WAV, независимо от расширения имени файла, но с этим методом можно ассоциировать звуковые файлы других форматов при помощи метода addSpeech(), например MP3. MP3-файлы создаются не методом synthesizeToFile() движка TTS, а несколько иначе.

Существует не очень много способов применения этого метода при синтезе речи. При использовании сценария с неограниченным количеством слов (то есть когда вы не знаете заранее, какое слово будет сказано следующим) невозможно иметь наготове аудиофайлы на все случаи и понадобится исправлять те слова, которые Pico будет произносить неправильно. Если набор слов ограничен (например, при зачитывании прогноза погоды), вам придется протестировать в программе все нужные слова, найти те, которые звучат неправильно, и подправить их. Даже в си-

туации с неограниченным набором слов целесообразно заранее записать некоторые слова, чтобы самые важные слова звучали правильно. Например, вам может понадобиться готовый звуковой файл с названием вашей фирмы либо с собственным именем.

Однако у метода с записью звуковых файлов есть и недостатки: текст, передаваемый в `speak()`, должен точно совпадать с текстом, используемым при вызове `addSpeech()`. К сожалению, невозможно записать аудиофайл для отдельно взятого слова, а затем ожидать, что TTS обязательно воспроизведет это слово из аудиофайла, если вы передадите это слово методу `speak()` в какой-либо фразе. Чтобы система воспроизвела аудиофайл, текст, передаваемый методу `speak()`, должен точно совпадать с текстом этого файла. Если во фразе будет чуть больше или чуть меньше информации, Pico не сработает, выше головы ей не прыгнуть. Один из способов обойти это неудобство — разбить текст на слова, и передать TTS каждое слово отдельно. Хотя при этом будут воспроизводиться только нужные нам файлы (например, сначала Don, потом Quixote), речь в результате получится прерывистой, поскольку каждое слово, по существу, будет отдельной фразой. В некоторых программах с этим можно мириться. Идеальный случай для применения аудиофайлов — это ситуация, в которой устройство должно произносить заранее заданные записанные слова и фразы, причем текст, который должен воспроизводиться, сразу нам известен.

Итак, что делать, если мы должны строить при помощи Pico предложения из слов и некоторые из них наверняка будут произнесены неправильно? Один из способов — просмотреть текст, узнать, нет ли в нем проблемных слов, и заменить их «искусственными», которые Pico сможет воспроизвести правильно. Например, перед вызовом `speak()` мы, вероятно, можем заменить Quixote на Kikhot. Результат нас устроит, а пользователь никак не заметит подмены. Что касается использования ресурсов, то сохранять «искусственные» слова гораздо целесообразнее, чем записывать аудиофайлы, пусть мы в данном случае и работаем только с Pico. Pico будет воспроизводить и весь оставшийся текст, то есть мы практически ничего не теряем. С другой стороны, не хотелось бы, чтобы Pico слишком много «домысливала». То есть Pico достаточно интеллектуальна, чтобы понимать, как произносятся отдельные слова, и если мы попытаемся подсказывать ей, то легко можем ее запутать и столкнуться с проблемами.

В нашем последнем примере мы записали звуковой файл для текстового фрагмента, то есть когда позже TTS будет должен нам его зачитать, он обратится к звуковому файлу, а не будет генерировать речь при помощи Pico. Очевидно, что на воспроизведение небольшого аудиофайла тратится меньше ресурсов, чем на запуск движка TTS и взаимодействие с ним. Следовательно, если у вас есть контролируемый список фраз или слов, предназначенных для произнесения, нужные аудиофайлы можно заготовить заранее, даже если Pico произносит их правильно. Так вы ускорите работу программы. Если у вас немного звуковых файлов, то в целом и памяти вы будете использовать мало. Применяя такой подход, вызываем метод так:

```
TextToSpeech.addSpeech(String text, String packagename,
        int soundFileResourceId)
```

Это очень простой способ добавления звуковых файлов к TTS. Текстовый аргумент — это строка, воспроизводимая в звуковом файле; packagename — название пакета того приложения, в котором сохранен файл ресурса, а soundFileResourceId — это ID ресурса звукового файла. Сохраняйте звуковые файлы в каталоге /res/raw. Когда ваша программа запускается, добавьте заранее записанные звуковые файлы к TTS, ссылаясь на их ID ресурса (например, R.raw.quixote). Конечно, здесь не обойтись без своего рода базы данных либо заранее заданного списка, в котором будет указано, какому тексту соответствует какой звуковой файл. Если вы создаете несколько локализованных вариантов приложения, то альтернативные звуковые файлы должны храниться в соответствующем каталоге /res/raw; например /res/raw-fr — для франкоязычных звуковых файлов.

Продвинутые функции синтезатора речи (TTS Engine)

Теперь, когда мы изучили основы преобразования текста в речь, рассмотрим некоторые более сложные функции TTS Engine. Сначала изучим аудиопотоки (audio streams), которые позволяют направлять произносимый текст в нужный аудиовыход. Далее поговорим о проигрывании звуковых пиктограмм (элементарных звуковых знаков, которые еще называются earcons) и о воспроизведении тишины. Наконец, рассмотрим, как устанавливаются языковые параметры, и в завершение обсудим вызовы различных методов.

Настройка потоков аудио

Выше мы использовали параметр HashMap для передачи движку TTS дополнительных аргументов. Один из таких аргументов (KEY_PARAM_STREAM) сообщает TTS, какой аудиопоток использовать для воспроизведения текста. В табл. 15.1 перечислены все доступные потоки аудио.

Таблица 15.1. Доступные аудиопотоки

Аудиопоток	Описание
STREAM_ALARM	Аудиопоток для тревожных сигналов
STREAM_MUSIC	Аудиопоток для воспроизведения музыки
STREAM_NOTIFICATION	Аудиопоток для уведомлений
STREAM_RING	Аудиопоток для телефонного звонка
STREAM_SYSTEM	Аудиопоток для системных звуков
STREAM_VOICE_CALL	Аудиопоток для голосовых вызовов

Если текст, который следует произнести, является сигналом тревоги, то мы приказываем TTS воспроизвести его в соответствующем потоке. Следовательно, мы не будем делать вызов при помощи следующего кода, перед тем как вызывать метод speak():

```
params.put(TextToSpeech.Engine.KEY_PARAM_STREAM,
                    String.valueOf(AudioManager.STREAM_ALARM));
```

Еще раз просмотрите листинг 15.2 и вспомните, как мы настраивали и передавали параметры HashMap при вызове метода speak(). Можно передавать ID речевых фрагментов в том же HashMap, в котором указывается аудиопоток.

Использование звуковых пиктограмм

Звуковые пиктограммы (earcons) — это особый тип звуков, которые может воспроизводить TTS. Эта пиктограмма напоминает значок с рабочего стола, так как имеет фиксированное значение, только мы ее не видим, а слышим. Она предназначена не для воспроизведения текста, а скорее для того, чтобы давать аудиоподсказку о том, что должно произойти то или иное событие, о том, что в тексте присутствует определенный элемент, который сам не является текстовым. Звуковая пиктограмма может, например, сигнализировать, что в данный момент мы читаем маркированный список из презентации либо что перелистывается страница. Может быть, ваша программа предназначена для сопровождения пешей экскурсии, и звуковой сигнал сообщает слушателю, что он может перейти к следующей части экскурсии.

Чтобы задать пиктограмму для воспроизведения, нужно активировать метод addEarcon(), принимающий два или три аргумента, примерно так же работает addSpeech(). Первый аргумент — это название звуковой пиктограммы, он похож на текстовое поле в addSpeech(). Принято заключать название звуковой пиктограммы в квадратные скобки, например [boing]. В случае с двумя аргументами второй аргумент — это строка, представляющая собой имя файла. Если аргумента три, то вторым аргументом будет имя пакета, а третьим — ID ресурса, относящийся к аудиофайлу, который, как правило, хранится в /res/raw. Чтобы воспроизвести звуковую пиктограмму, используйте метод playEarcon(), который выглядит почти как метод speak() с тремя его аргументами.

Причины, по которым мы используем звуковые пиктограммы, а не просто воспроизводим аудиофайлы при помощи MediaPlayer, связаны с действующим в TTS механизмом ведения очередей (queuing mechanism). Вместо того чтобы определять удобный момент для воспроизведения звуковой подсказки (в таком случае нам придется полагаться на обратные вызовы, чтобы обеспечить правильный хронометраж), можно поставить звуковые пиктограммы в той же очереди, в которой у нас стоят текстовые фрагменты, отправляемые к TTS. Так мы гарантируем, что пользователь услышит аудиознаки именно тогда, когда нужно, а для воспроизведения звука мы можем использовать тот же путь, что и для отображения текста. В том числе у нас в арсенале остаются обратные вызовы onUtteranceCompleted(), позволяющие понять, на каком этапе мы находимся.

Воспроизведение тишины

В TTS имеется еще один метод воспроизведения, с которым мы можем работать, — playSilence(). Этот метод, подобно speak() и playEarcon(), также может иметь три аргумента, причем второй аргумент — это способ построения очередей, а третий — необязательный аргумент params HashMap. Первый аргумент метода playSilence() — это число типа long, указывающее, в течение скольких миллисекунд должна

воспроизводиться тишина. Обычно этот метод используется в режиме QUEUE_ADD, чтобы отделить друг от друга воспроизведение двух текстовых строк. Это означает, что вы можете вставить фрагмент тишины между двумя строками текста, не управляя продолжительностью ожидания прямо в программе. Мы просто вызываем speak(), потом playSilence(), а затем делаем еще один вызов speak(), чтобы получить нужный эффект.

Использование языковых методов

У нас остались нерассмотренными еще языковые вопросы, и сейчас мы ими займемся. Механизм синтеза речи воспроизводит текст на том языке, на котором этот текст написан. Например, итальянский текст должен читаться с итальянским произношением. Чтобы произносить текст правильно, система ищет в нем характерные особенности. Поэтому нельзя использовать неверное произношение с текстом, отправляемым к TTS. Если произнести французский текст голосом итальянца, в системе могут возникнуть проблемы. Лучше, чтобы языковой аспект текста и голоса совпадали (то есть чтобы были одинаковыми язык, на котором произносится текст, и выговор, характерный для этого языка).

Движок TTS предоставляет методы для работы с языками: и для того, чтобы узнать, какие языки доступны, и для установки языка воспроизведения. В TTS предусмотрен лишь ограниченный набор языковых пакетов, хотя всегда можно выйти на Android Market и достать другие пакеты, если они есть. Немного подобного кода присутствует в листинге 15.1, в обратном вызове onActivityResult(), где намерение (Intent) создавалось для получения недостающего языка. Разумеется, нужный языковой пакет может отсутствовать, но со временем доступных пакетов будет все больше.

Метод для проверки языка — это isLanguageAvailable(Locale locale). Поскольку локальные данные могут представлять и страну, и язык, а иногда просто вариант, возвращаемый ответ не ограничен значениями true или false. Возможны следующие варианты ответа:

- TextToSpeech.LANG_COUNTRY_AVAILABLE — поддерживаются и страна, и язык;
- TextToSpeech.LANG_AVAILABLE — поддерживается только язык, но не страна;
- TextToSpeech.LANG_NOT_SUPPORTED — не поддерживается ни то, ни другое.

Если вы получаете в ответ TextToSpeech.LANG_MISSING_DATA, это означает, что язык поддерживается, но TTS не удалось найти информационные файлы. Тогда ваша программа должна направить пользователя на Android Market или другой подходящий ресурс, где можно найти недостающие файлы. Например, французский язык может поддерживаться, а его канадский вариант — нет. Если мы имеем дело с таким случаем и TTS передается Locale.CANADA_FRENCH, то ответ будет TextToSpeech.LANG_AVAILABLE, а не TextToSpeech.LANG_COUNTRY_AVAILABLE. Есть еще один особый случай, когда LOCALE может включать региональный вариант. Тогда ответ будет таким: TextToSpeech.LANG_COUNTRY_VAR_AVAILABLE. Это значит, что поддерживается все.

Для установки языка используется метод setLanguage(Locale locale). В результате мы получаем такие же коды, как и от isLanguageAvailable(). Чтобы получить

актуальные локализационные данные, действующие на устройстве по умолчанию, примените метод Locale.getDefault(), который возвратит нужное локальное значение, например en_US, либо другое, подходящее для вашего местоположения. Используйте метод getLanguage() класса TextToSpeech, чтобы выяснить действующие локализационные данные TTS. Нам было бы очень кстати реализовать что-то подобное в нашем предыдущем примере, чтобы задать язык для TTS:

```
switch(mTts.setLanguage(Locale.getDefault())) {
case TextToSpeech.LANG_COUNTRY_AVAILABLE: …
```

Наконец, чтобы завершить разговор о синтезе речи, поговорим еще о нескольких методах, которые могут использоваться в этом контексте. Метод setPitch(float pitch) изменяет высоту голоса, не изменяя его скорости. Стандартное значение высоты — 1.0. Самая низкая значащая величина, по-видимому, равна 0.5, а самая высокая — 2.0. Можно устанавливать более высокие и более низкие значения, но после пересечения этих порогов изменение высоты голоса практически не ощущается на слух. Скорее всего, такие же пороги действуют для метода setSpeechRate(float rate). Это значит, что вы передаете данному методу аргумент, являющийся числом с плавающей точкой, в диапазоне от 0.5 до 2.0, где нормальный темп речи будет равен 1.0. Значение выше 1.0 означает быструю речь, ниже 1.0 — медленную речь. Еще один метод, который может вам пригодиться, — isSpeaking(). Он возвращает значения true или false, чтобы указать, произносит ли TTS в данный момент что-либо (в том числе тишину, метод playSilence()). Если требуется уведомление о том, что TTS закончил воспроизводить все речевые фрагменты, стоявшие в очереди, можно внедрить BroadcastReceiver для ACTION_TTS_QUEUE_PROCESSING_COMPLETED.

Перевод текста на другой язык

В первой части этой главы мы рассмотрели, как воспроизвести текст, чтобы пользователь мог его прослушать. Мы научились задавать акцент голоса при помощи Text to Speech. А в этом разделе будет показано, как переводить текст с одного языка на другой. Вооружившись Text to Speech, вы сможете брать текст на одном языке и зачитывать его пользователю на другом, с правильным акцентом.

Перевод с языка на язык — задача не из тех, которые хорошо решаются на мобильном устройстве. В одном только английском языке насчитывается несколько сотен тысяч слов, а возможно, и миллион (смотря как понимать границы английского языка). Задача загрузки в мобильное устройство языков и правил просто для того, чтобы осуществлять перевод между двумя любыми языками, практически невыполнима.

В Интернете Google имеет API, выполняющий переводы. Система берет текстовую строку, пару языковых спецификаций (язык оригинала и язык перевода) и переводит текст с одного языка на другой. Но здесь есть особенность. Первоначально эта служба задумывалась для работы на веб-сайтах, а не на мобильных устройствах. В условиях использования интерфейса Google AJAX Language API (так он официально называется) не предусмотрен вариант для работы с мобильными устройствами, в отличие, например, от Google Maps API. Условия использования

интерфейса AJAX Language API даны на следующем сайте: http://code.google.com/apis/ajaxlanguage/terms.html.

Хотя Google прямо и не указывает, что разработчики под Android должны использовать этот API, на конференции Google I/O в мае 2009 года данный API демонстрировался именно на примере программы для Android! Возможно, пока вы читаете эти строки, Google разрабатывает отдельные условия использования AJAX Language API с Android, либо вносятся изменения в имеющиеся условия, чтобы было понятно, что они предназначены и для работы с Android. Пока у вас есть два выхода. Во-первых, можете смело приступать к делу и использовать AJAX Language API прямо в вашей программе Android; этот подход будет объяснен ниже. Во-вторых, можно организовать доступ к AJAX Language API через управляемый вами веб-сервер, например прокси-сервер. Ваша программа будет иметь «стык» с вашим же веб-сервером, а ваш веб-сервер будет делать вызовы AJAX Language API. Имея в центре системы подконтрольный веб-сервер, гораздо проще будет деактивировать доступ программы к AJAX Language API, так как вы контролируете соединяющий их «перевал». Разумеется, ситуация такова, что, если у вас исчезнет доступ к службе Google, вы практически не сможете работать с вашей программой. Как минимум в программе можно реализовать специфический вид отклика, означающий, что служба Google временно недоступна, и выдать пользователю соответствующее сообщение. В первом случае, если Google потребует прекратить использование AJAX Language API, вы ничего не сможете поделать. Ваша программа уже будет распространена на множестве устройств, и если вы не обеспечите какой-либо путь, позволяющий работать без этого API, устройства снова и снова будут пытаться получить к нему доступ.

Google вправе отказать вам в таком доступе, но сделать это практически, возможно, будет непросто. В условиях использования Google не указывает, что для работы с AJAX Language API обязательно нужен ключ от этого API, хотя в документации разработчика (http://code.google.com/apis/ajaxlanguage/documentation) сказано, что разработчик *обязан* использовать РЕФЕРЕР и ему *следует* использовать ключ от API. Без этих компонентов ваши запросы с пользовательских устройств будут анонимными и Google не сможет с вами связаться, если в вашей программе возникнут сложности с использованием API. В нашем следующем примере мы решили задать значение заголовка `REFERER` (см. код `Translator.java`), но опустили часть, в которой содержится ключ API. Если вы хотите послать значение ключа API к AJAX Language API, сначала понадобится получить такой ключ от Google. На выбор предоставляется несколько ключей API, но когда мы писали эту главу, еще не было способа получить ключ специально для AJAX Language API. Ключи к AJAX Language API выдаются вместе с их собственными условиями использования, с которыми вы должны согласиться прежде, чем получите новый ключ к API. Обратите внимание — ключ к Maps API не подходит к AJAX API. Чтобы зарегистрировать ключ к AJAX API, нужно просто отправить URL вашего сайта (ту самую, которую вы использовали как `REFERER`) и согласиться с условиями использования. Добавьте новый ключ API к AJAX API URL при помощи следующего кода:

```
&key=Your_API_key_goes_here_with_no_quotation_marks (здесь_вы_указываете_ключ_без_
всяких_кавычек)
```

Если вы решите передать ключ API к AJAX API, то в качестве REFERER нужно задать ту же URL, что использовалась при создании ключа к API, либо любую подстраницу этой URL. Иначе результата вы не получите.

В оставшейся части главы мы создадим приложение, которое будет напрямую вызывать Google AJAX Language API. До этого момента на страницах нашей книги уже были описаны все компоненты, необходимые для того, чтобы оснастить программу функцией перевода. Теперь мы соберем их вместе. Например, мы создаем приложение с полем EditText, предназначенным для ввода, используем блоки с изменяемыми значениями (spinners) для выбора языка оригинала и перевода; EditText, предназначенный только для чтения, в котором выводится перевод; активируем службу через Интернет. А также используем службу, позволяющую изолировать пользовательский интерфейс от логики, на выполнение которой может уйти какое-то время. Один из дополнительных компонентов, который понадобится включить в программу, — проект Jakarta Commons Lang, предназначенный для того, чтобы декодировать (unescape) символы XML в Unicode — так, чтобы они могли отображаться. Мы объясним, как это делается. XML-шаблон приведен в листинге 15.5, а на рис. 15.4 показано, как это выглядит на экране. В последующих листингах мы «препарируем» всю программу. Хотя на первый взгляд и может показаться, что эта глава переполнена кодом, — поверьте, его совсем не много, учитывая, какие функции он обеспечивает. (Кроме того, все детали этого примера уже встречались в предыдущих главах, начиная от настройки службы и заканчивая применением блоков с изменяемыми значениями как элементов раскрывающегося списка.)

Листинг 15.5. XML-шаблон, применяемый при внедрении машинного перевода

```
<?xml version="1.0" encoding="utf-8"?>
<!--Это файл /res/layout/main.xml -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="vertical"
 android:layout_height="fill_parent"
 android:layout_width="fill_parent">

     <EditText android:id="@+id/input"
 android:hint="@string/input"
 android:layout_height="wrap_content"
 android:layout_width="fill_parent" />

     <Spinner android:id="@+id/from"
         android:layout_weight="1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_below="@id/input"
         android:prompt="@string/prompt" />

     <Button android:id="@+id/translateBtn"
         android:text="@string/translateBtn"
         android:layout_weight="1"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
```

```
        android:layout_below="@id/input"
        android:layout_toRightOf="@id/from"
        android:enabled="false" />

    <Spinner android:id="@+id/to"
        android:layout_weight="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/input"
        android:layout_toRightOf="@id/translateBtn"
        android:prompt="@string/prompt" />

 <EditText android:id="@+id/translation"
     android:hint="@string/translation"
     android:layout_height="wrap_content"
     android:layout_width="fill_parent"
     android:editable="false"
     android:layout_below="@id/from" />

 <TextView android:id="@+id/poweredBy"
     android:text="powered by Google"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_alignParentBottom="true" />

</RelativeLayout>
```



Рис. 15.4. Пользовательский интерфейс демонстрационной программы для перевода

Шаблон очень прост. Мы задаем пары полей для текста — соответственно для фразы в оригинале и в переводе. Кроме того, указываем блоки с изменяемыми значениями в виде раскрывающихся меню для выбора языков оригинала и перевода. Нам еще понадобится кнопка для запуска перевода и, наконец, строка powered

by Google (При поддержке Google), располагаемая внизу экрана (ниже мы подробнее объясним, зачем нужна эта строка). В листинге 15.6 приведены файлы `strings.xml` и `arrays.xml`, в которых задаются строки из нашего пользовательского интерфейса, а также описываются меню.

Листинг 15.6. strings.xml и arrays.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/values/strings.xml -->
<resources>
   <string name="translateBtn">> Translate ></string>
   <string name="input">Enter the text to translate</string>
   <string name="translation">The translation will appear here</string>
   <string name="prompt">Choose a language</string>
</resources>

<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/values/arrays.xml -->
<resources>
<string-array name="languages">
   <item>Chinese</item>
   <item>English</item>
   <item>French</item>
   <item>German</item>
   <item>Japanese</item>
   <item>Spanish</item>
</string-array>
<string-array name="language_values">
   <item>zh</item>
   <item>en</item>
   <item>fr</item>
   <item>de</item>
   <item>ja</item>
   <item>es</item>
</string-array>
</resources>
```

Теперь, когда наш пользовательский интерфейс в целом готов, создадим службу, которая будет обеспечивать взаимодействие с Google AJAX Language API. В листинге 15.7 приведены файлы, определяющие служебный интерфейс.

Листинг 15.7. Файлы служебного интерфейса для программы-переводчика

```
// это файл ITranslate.aidl из каталога /src
interface ITranslate {
   String translate(in String text, in String from, in String to);
}

// это файл TranslateService.java
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
```

```
import android.util.Log;

public class TranslateService extends Service {
   public static final String TAG = "TranslateService";

   private final ITranslate.Stub mBinder = new ITranslate.Stub() {
      public String translate(String text, String from, String to) {
         try {
            return Translator.translate(text, from, to);
         } catch (Exception e) {
            Log.e(TAG, "Failed to translate: " + e.getMessage());
            return null;
         }
      }
   };

   @Override
   public IBinder onBind(Intent intent) {
      return mBinder;
   }
}
```

Не забывайте о том, что Eclipse автоматически допишет код Java на основе файла AIDL, как только мы сохраним этот файл в Eclipse. Наша новая служба должна инициировать статический метод `Translator.translate()`, который содержится в коде из листинга 15.8.

Листинг 15.8. Код Java для взаимодействия с Google AJAX Language API

```
// это файл Translator.java
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

import org.apache.commons.lang.StringEscapeUtils;
import org.json.JSONObject;
import android.util.Log;

public class Translator {
   private static final String ENCODING = "UTF-8";
   private static final String URL_BASE = ➥
"http://ajax.googleapis.com/ajax/services/language
/translate?v=1.0&langpair=";
   private static final String INPUT_TEXT = "&q=";
   private static final String MY_SITE = "http://my.website.com";
   private static final String TAG = "Translator";

   public static String translate(String text,
            String from, String to) throws Exception
{
```

```
        try {
            StringBuilder url = new StringBuilder();
            url.append(URL_BASE).append(from).append("%7C").append(to);
            url.append(INPUT_TEXT).append(
                    URLEncoder.encode(text, ENCODING));

            HttpURLConnection conn = (HttpURLConnection) new
                    URL(url.toString())
                    .openConnection();
            conn.setRequestProperty("REFERER", MY_SITE);
            conn.setDoInput(true);
            conn.setDoOutput(true);
            try {
                InputStream is= conn.getInputStream();
                String rawResult = makeResult(is);

                JSONObject json = new JSONObject(rawResult);
                String result = ((JSONObject)json.get("responseData"))
                                      .getString("translatedText");
                return (StringEscapeUtils.unescapeXml(result));
            } finally {
                conn.getInputStream().close();
                if(conn.getErrorStream() != null)
                    conn.getErrorStream().close();
            }
        } catch (Exception ex) {
            throw ex;
        }
    }

    private static String makeResult(InputStream inputStream)
                throws Exception {
        StringBuilder outputString = new StringBuilder();
        try {
            String string;
            if (inputStream != null) {
                BufferedReader reader =
                    new BufferedReader(new InputStreamReader(inputStream,
                        ENCODING));
                while (null != (string = reader.readLine())) {
                        outputString.append(string).append('\n');
                }
            }
        } catch (Exception ex) {
            Log.e(TAG, "Error reading translation stream.", ex);
        }
        return outputString.toString();
    }
}
```

В классе Translator содержится вся изюминка нашего демонстрационного приложения. В принципе, эта программа делает HTTP-вызов к службе Google AJAX

Language API, а затем считывает ответ. Несколько позже мы рассмотрим этот механизм в подробностях, но сначала закончим наше демонстрационное приложение — чтобы его можно было протестировать. В листинге 15.9 приведен код Java для явления MainActivity.

Листинг 15.9. Код Java для MainActivity

```
// это файл MainActivity.java
import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

public class MainActivity extends Activity implements OnClickListener {
   static final String TAG = "Translator";
   private EditText inputText = null;
   private TextView outputText = null;
   private Spinner fromLang = null;
   private Spinner toLang = null;
   private Button translateBtn = null;
   private String[] langShortNames = null;
   private Handler mHandler = new Handler();

   private ITranslate mTranslateService;

   private ServiceConnection mTranslateConn = new ServiceConnection() {
      public void onServiceConnected(ComponentName name,
                              IBinder service) {
         mTranslateService = ITranslate.Stub.asInterface(service);
         if (mTranslateService != null) {
            translateBtn.setEnabled(true);
         } else {
            translateBtn.setEnabled(false);
            Log.e(TAG, "Unable to acquire TranslateService");
         }
      }

      public void onServiceDisconnected(ComponentName name) {
         translateBtn.setEnabled(false);
```

```
            mTranslateService = null;
        }
    };

    @Override
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        inputText = (EditText) findViewById(R.id.input);
        outputText = (EditText) findViewById(R.id.translation);
        fromLang = (Spinner) findViewById(R.id.from);
        toLang = (Spinner) findViewById(R.id.to);

        langShortNames = getResources().getStringArray(
                          R.array.language_values);

        translateBtn = (Button) findViewById(R.id.translateBtn);
        translateBtn.setOnClickListener(this);

        ArrayAdapter<?> fromAdapter = ArrayAdapter.createFromResource(this,
                R.array.languages, android.R.layout.simple_spinner_item);

        fromAdapter.setDropDownViewResource(
                android.R.layout.simple_dropdown_item_1line);
        fromLang.setAdapter(fromAdapter);
        fromLang.setSelection(1); // английский

        ArrayAdapter<?> toAdapter = ArrayAdapter.createFromResource(this,
                R.array.languages,android.R.layout.simple_spinner_item);
        toAdapter.setDropDownViewResource(
                android.R.layout.simple_dropdown_item_1line);
        toLang.setAdapter(toAdapter);
        toLang.setSelection(3); // немецкий

        inputText.selectAll();

        Intent intent = new Intent(Intent.ACTION_VIEW);
        bindService(intent, mTranslateConn, Context.BIND_AUTO_CREATE);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unbindService(mTranslateConn);
    }

    public void onClick(View v) {
        if (inputText.getText().length() > 0) {
            doTranslate();
        }
    }
```

```
    private void doTranslate() {
        mHandler.post(new Runnable() {
            public void run() {
                String result = "";
                try {
                    int fromPosition = fromLang.getSelectedItemPosition();
                    int toPosition = toLang.getSelectedItemPosition();
                    String input = inputText.getText().toString();
                    if(input.length() > 5000)
                        input = input.substring(0,5000);
                    Log.v(TAG,"Translating from " +
                            langShortNames[fromPosition] + " to " +
                            langShortNames[toPosition]);
                    result = mTranslateService.translate(input,
                            langShortNames[fromPosition],
                            langShortNames[toPosition]);
                    if (result == null) {
                        throw new Exception("Failed to get a translation");
                    }
                    outputText.setText(result);
                    inputText.selectAll();
                } catch (Exception e) {
                    Log.e(TAG, "Error: " + e.getMessage());
                }
            }
        });
    }
}
```

В нашем явлении MainActivity настраиваются пользовательский интерфейс и служба, а затем предоставляется метод для вызова, запускающего процесс перевода после нажатия кнопки. Осталось сделать ровно одну вещь — сконфигурировать файл AndroidManifest.xml, приведенный в листинге 15.10. Обратите внимание — требуется право для доступа в Интернет, чтобы активировать Google AJAX Language API.

Листинг 15.10. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл AndroidManifest.xml -->
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.androidbook.translation"
      android:versionName="1.0"
      android:versionCode="1" >

    <application android:label="Translate"
        android:icon="@drawable/icon">

        <activity android:name="MainActivity" android:label="Translate">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <service android:name="TranslateService" android:label="Translate">
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
    </service>
  </application>
  <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

Прежде чем этот пример будет завершен, нужно сделать вспомогательный класс. В проекте Jakarta Commons Lang есть класс StringEscapeUtils, который можно использовать для преобразования строки результатов, полученной от AJAX Language API, во что-нибудь удобочитаемое. AJAX Language API способен возвращать сущности XML, соответствующие конкретным специальным символам. Например, апостроф выражается как '. Эти специальные символы необходимо правильно отобразить пользователю. Именно в этом вам и поможет проект Jakarta Commons Lang. Скачать его можно здесь: **http://commons.apache.org/lang/**.

Перейдите на сайт Jakarta Commons Lang, найдите и скачайте нужный файл ZIP (для Windows) или TAR (для Mac OS X или Linux), в котором содержится файл JAR. Распакуйте этот файл JAR — он понадобится на следующем этапе. В Eclipse необходимо выбрать проект, щелкнуть на нем правой кнопкой мыши и выполнить **Build Path ▸ Configure Build Path** (Создать путь ▸ Конфигурировать созданный путь). Откройте вкладку **Libraries** (Библиотеки), в ней выберите **Add external JARs** (Добавить внешние файлы JAR). Перейдите к JAR-файлу common-lang и добавьте его. Теперь нажмите **OK**, чтобы закрепить файл в проекте. Программа должна достроиться до полной готовности. Попробуйте с ней поработать. Если окно программы не умещается на экране в «книжном» формате, нажмите **Ctrl+F12** и переведите эмулятор в альбомный формат. Если полученные результаты вызывают у вас сомнения, сравните их с результатами Google: **http://www.google.com/uds/samples/language/translate.html**.

Есть еще элементы, о которых следует поговорить отдельно. В соответствии с условиями использования мы включили в пользовательский интерфейс строку **powered by Google** (При поддержке Google). Кроме того, в условиях использования указано, что строка не должна превышать в размере 5000 символов, поэтому лишние символы придется отсекать. Возможен и другой выход, например разбить текст на управляемые фрагменты, которые затем можно передавать к API — так ничего не потеряется. Мы специально оставили список языков кратким, чтобы программой было проще управлять, но вы можете свободно добавлять новые языки в строковые массивы, чтобы получались дополнительные языковые пары. Однако учитывайте, что в шрифтах Droid может не найтись символов для всех языков, с которыми будет работать переводчик. Шрифты Droid были разработаны специально для Android, но набор символов в них ограничен. Если в результатах будут какие-нибудь

ошибки со шрифтами, то, чтобы исправить их, попробуйте найти дополнительные. В этой главе мы не будем подробно говорить о шрифтах. Ответ, получаемый от API, структурируется с применением нотации JSON. Затем JSON применяется для синтаксического разбора полученного ответа в результирующую строку. (Обратите внимание: JSON в данном случае предоставляется как часть системы Android, поэтому мы можем не брать его из Интернета и не подключать как внешний файл JAR.)

Одна из характерных особенностей AJAX Language API заключается в том, что при работе с ним нет необходимости указывать язык ввода (input language). API попытается угадать его. Если вы решите применить такой метод, то можете не задавать язык ввода в передаваемой URL, а поставить `%7C` сразу после `langpair=`. Это наилучший метод, если вы не знаете точно, с каким языком придется работать. Однако, чтобы API выбрал язык правильно, ему нужно передать достаточно большой объем текста.

Резюме

В этой главе мы рассказали, как заставить программу Android говорить с пользователем. Для выполнения этой функции в Android встроен очень симпатичный движок TTS. Разработчик без труда поймет, как с ним обращаться. Система Pico выполняет за нас большую часть работы. Если с Pico возникают проблемы, все равно остаются способы достичь желаемого эффекта — они описаны выше. Продвинутые функции также сильно облегчают работу. Используя синтезатор речи, нужно всегда помнить о том, что он установлен в мобильном устройстве: экономьте ресурсы, организуйте совместное применение TTS и правильно задействуйте голос.

Мы также показали, как обращаться к Google API через Интернет. В нашем конкретном случае мы пользовались Google AJAX Language API, но та же самая техника открывает доступ и к другим API Google. Итак, если вам нужен мгновенный перевод с одного языка на другой — вы уже знаете, как это сделать.

16 Сенсорные экраны

Во многих устройствах Android имеются сенсорные экраны. Если на устройстве нет физической клавиатуры, значительная часть пользовательского ввода *может* поступать только через сенсорный экран. Поэтому вашим программам зачастую придется работать с сенсорным вводом данных. Вы, наверное, уже обращали внимание, что, когда от пользователя требуется ввести текст, на экране появляется виртуальная клавиатура. Сенсорный ввод применялся при работе с картографическими программами в главе 7 при панорамировании карт для взгляда «сбоку». До сих пор мы не указывали на то, что в том или ином примере задействован сенсорный интерфейс, но теперь рассмотрим, как пользоваться его преимуществами.

Эта глава состоит из четырех основных разделов. В первом мы научимся работать с объектами `MotionEvents`, при помощи которых Android сообщает программе, что пользователь касается экрана. Во втором мы рассмотрим технологию мультитач, при которой пользователь касается экрана несколькими пальцами одновременно. В третьем разделе рассказано, какую роль играет сенсорный ввод при работе с картами, поскольку в Android имеются специальные методы и классы, предназначенные для работы с картами и сенсорными экранами. Наконец, мы поговорим о жестах, особом типе возможностей, позволяющих интерпретировать серии касаний как команды.

Понятие о MotionEvents

В этом разделе будет рассмотрено, как Android сообщает программе, что пользователь касается экрана. Пока мы изучим лишь случаи, при которых пользователь касается только одним пальцем (а мультитач рассмотрим в следующем разделе).

На аппаратном уровне сенсорный экран изготавливается из специальных материалов, которые могут воспринимать давление и преобразовывать его в координаты одной из точек экрана. Информация о касании преобразуется в данные, которые передаются программе для обработки.

Когда пользователь касается сенсорного экрана устройства Android, создается объект `MotionEvent`. В нем содержится информация о том, в какой точке и когда произошло касание, а также другие подробности события касания (touch event).

Объект MotionEvent передается соответствующему методу вашей программы. Например, это может быть метод onTouchEvent() объекта View. Как вы помните, класс View является родительским для некоторых других классов Android, в том числе Layout, Button, List, Surface, Clock и др. Таким образом, можно взаимодействовать со всеми, самыми разными, типами объектов View при помощи событий касания. При вызове такой метод может проверить объект MotionEvent и решить, что делать. Например, MapView может использовать события касания, чтобы двигать карту в режиме «вид сбоку» и позволять пользователю панорамировать интересующие его точки. Либо события касания могут восприниматься виртуальной клавиатурой, в результате чего будут активироваться виртуальные клавиши и в определенной части пользовательского интерфейса (UI) будет выполняться текстовый ввод.

Объект MotionEvent относится к последовательности событий, происходящих, когда пользователь касается экрана. Последовательность начинается при прикосновении и заканчивается, когда палец отрывается от сенсорного экрана. Первое прикосновение (действие ACTION_DOWN), движения в стороны (действие ACTION_MOVE) и снятие пальца с экрана (up event; действие ACTION_UP) — все эти события создают объекты MotionEvent. Что касается событий ACTION_MOVE, их может быть достаточно много, пока палец будет двигаться по экрану и до того момента, как произойдет ACTION_UP. В каждом объекте содержится информация о том, какое действие при этом выполняется, где произошло прикосновение, насколько велико было давление, размер площади прикосновения в момент действия, в также когда именно произошло первое прикосновение (ACTION_DOWN). Четвертое возможное действие — ACTION_CANCEL. Это действие используется в случае, когда прикосновение завершается без выполнения каких-либо операций. Наконец, существует действие ACTION_OUTSIDE, совершаемое в том особом случае, когда прикосновение происходит за пределами окна с программой.

Есть еще один способ получения информации о событиях касания — регистрация обработчика обратных вызовов с объектом View. В классе для получения событий необходимо реализовать интерфейс View.OnTouchListener, а метод setOnTouchListener() объекта View нужно вызывать, чтобы установить обработчик для этого View. В реализующем классе интерфейса View.OnTouchListener должен применяться метод onTouch(). В то время как метод onTouchEvent() принимает в качестве параметра только объект MotionEvent, метод onTouch() принимает в качестве параметров оба объекта — View и MotionEvent. Это объясняется тем, что OnTouchListener может получать объекты MotionEvent от нескольких видов. В следующем примере с программой этот момент объяснен более детально.

Если обработчик события MotionEvent (методом onTouchEvent() или onTouch()) выполняет действие над событием и об этом не нужно сообщать никакому иному элементу, метод должен возвратить значение true. Так Android узнает, что произошедшее событие не следует передавать никаким другим видам. Если конкретный объект View «не заинтересован» в этом событии, *а также во всех остальных событиях данной последовательности касаний*, метод возвращает false. Метод onTouchEvent() базового класса View не выполняет никаких действий и возвращает false. Субклассы View могут вести себя и так же, и по-другому. Например, объект Button воспримет касание, так как оно равнозначно нажатию (click) и, следовательно, вернет от метода onTouchEvent() значение true. При событии ACTION_DOWN объект

Button изменит цвет, чтобы указать, что произошло нажатие. Следовательно, данная кнопка должна также получить событие ACTION_UP, когда пользователь уберет палец, — так инициируется логика нажатия кнопки. Если бы объект Button вернул false от onTouchEvent(), то уже не получил бы объектов MotionEvent, которые сообщили бы ему, что пользователь убрал палец с экрана.

Если мы хотим, чтобы в конкретном виде касания инициировали какие-то иные события, мы можем расширить класс, переопределить метод onTouchEvent() и задать в нем нужную нам логику. Кроме того, можно реализовать здесь интерфейс View.OnTouchListener и установить обработчик обратных вызовов для объекта View. Если установить обработчик обратных вызовов с методом onTouch(), то MotionEvents будут доставляться к нему до того, как отправляться к методу onTouchEvent() объекта View. Наш метод onTouchEvent() объекта View в таком случае будет вызываться лишь при условии, что метод onTouch() вернет false. Перейдем к коду программы — на примере все будет понятнее.

В листинге 16.1 показан XML-шаблон. Создадим в Eclipse новый проект Android и начнем с этого шаблона.

Листинг 16.1. Файл XML-шаблона для TouchDemo1

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical" >

   <RelativeLayout
      android:id="@+id/layout1"
      android:tag="trueLayoutTop"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      >

      <com.androidbook.touch.demo1.TrueButton android:text="returns true"
      android:id="@+id/trueBtn1"
      android:tag="trueBtnTop"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />

      <com.androidbook.touch.demo1.FalseButton android:text=
                  "returns false"
      android:id="@+id/falseBtn1"
      android:tag="falseBtnTop"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_below="@id/trueBtn1" />

   </RelativeLayout>
   <RelativeLayout
```

```
        android:id="@+id/layout2"
        android:tag="falseLayoutBottom"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#FF00FF"
        >

        <com.androidbook.touch.demo1.TrueButton android:text="returns true"
        android:id="@+id/trueBtn2"
        android:tag="trueBtnBottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

        <com.androidbook.touch.demo1.FalseButton android:text=
                 "returns false"
        android:id="@+id/falseBtn2"
        android:tag="falseBtnBottom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/trueBtn2" />

    </RelativeLayout>
</LinearLayout>
```

В этом шаблоне необходимо отметить несколько моментов. С объектами нашего пользовательского интерфейса применяются теги. Мы сможем ссылаться на эти теги в нашем коде, когда с объектами будут происходить события. Кроме того, для расположения объектов использовались RelativeLayout. Обратите также внимание на то, как мы работаем с пользовательскими объектами (TrueButton и FalseButton). В коде Java видно, что эти классы являются производными от класса Button. На рис. 16.1 показано, как выглядит этот шаблон, а в листинге 16.2 приведен код Java для нашей кнопки.

Листинг 16.2. Код Java для классов из TouchDemo1, касающихся работы с кнопкой

```
// это файл BooleanButton.java
import android.content.Context;
import android.util.AttributeSet;
import android.util.Log;
import android.view.MotionEvent;
import android.widget.Button;

public abstract class BooleanButton extends Button {
   protected boolean myValue() {
      return false;
   }

   public BooleanButton(Context context, AttributeSet attrs) {
      super(context, attrs);
```

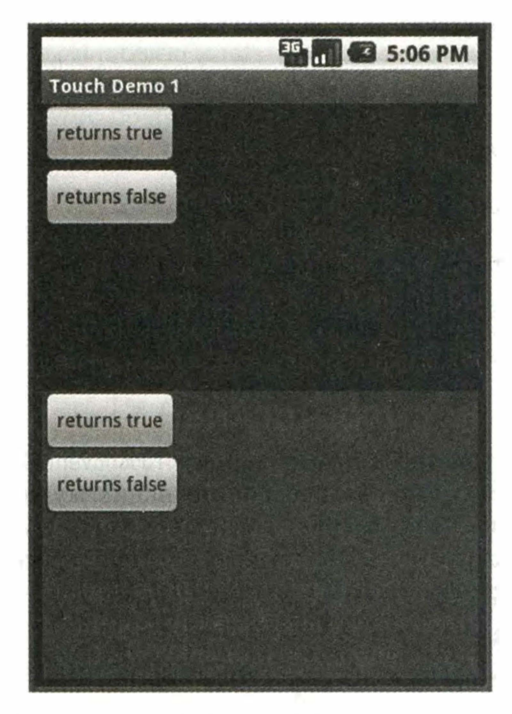


Рис. 16.1. Пользовательский интерфейс программы TouchDemo1

```
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        String myTag = this.getTag().toString();
        Log.v(myTag, "-----------------------------------");
        Log.v(myTag, MainActivity.describeEvent(this, event));
        Log.v(myTag, "super onTouchEvent() returns " +
                super.onTouchEvent(event));
        Log.v(myTag, "and I'm returning " + myValue());
        event.recycle();
        return(myValue());
    }
}

// это файл TrueButton.java
import android.content.Context;
import android.util.AttributeSet;

public class TrueButton extends BooleanButton {
    protected boolean myValue() {
        return true;
    }

    public TrueButton(Context context, AttributeSet attrs) {
```

```
      super(context, attrs);
   }
}

// это файл FalseButton.java
import android.content.Context;
import android.util.AttributeSet;

public class FalseButton extends BooleanButton {

   public FalseButton(Context context, AttributeSet attrs) {
      super(context, attrs);
   }
}
```

Класс BooleanButton построен так, что метод onTouchEvent() может использоваться многократно. К этому методу мы добавили функцию журналирования (logging). Затем создаются кнопки TrueButton и FalseButton, которые будут по-разному отвечать на передаваемые им MotionEvents. Этот момент будет понятнее после изучения кода основного явления, приведенного в листинге 16.3.

Листинг 16.3. Код Java для основного явления

```
// это файл MainActivity.java
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.Button;
import android.widget.RelativeLayout;

public class MainActivity extends Activity implements OnTouchListener {
   /** Вызывается при первом создании явления. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      RelativeLayout layout1 = (RelativeLayout)
                     findViewById(R.id.layout1);
      layout1.setOnTouchListener(this);
      Button trueBtn1 = (Button)findViewById(R.id.trueBtn1);
      trueBtn1.setOnTouchListener(this);
      Button falseBtn1 = (Button)findViewById(R.id.falseBtn1);
      falseBtn1.setOnTouchListener(this);

      RelativeLayout layout2 = (RelativeLayout)
                     findViewById(R.id.layout2);
      layout2.setOnTouchListener(this);
```

```
        Button trueBtn2 = (Button)findViewById(R.id.trueBtn2);
        trueBtn2.setOnTouchListener(this);
        Button falseBtn2 = (Button)findViewById(R.id.falseBtn2);
        falseBtn2.setOnTouchListener(this);
    }

    @Override
    public boolean onTouch(View v, MotionEvent event) {
        String myTag = v.getTag().toString();
        Log.v(myTag, "----------------------------");
        Log.v(myTag, "Got view " + myTag + " in onTouch");
        Log.v(myTag, describeEvent(v, event));
        if( "true".equals(myTag.substring(0, 4))) {
            Log.v(myTag, "and I'm returning true");
            return true;
        }
        else {
            Log.v(myTag, "and I'm returning false");
            return false;
        }
    }

    protected static String describeEvent(View view, MotionEvent event) {
        StringBuilder result = new StringBuilder(300);
        result.append("Action: ").append(event.getAction()).append("\n");
        result.append("Location: ").append(event.getX()).append(" x ")
                            .append(event.getY()).append("\n");
        if( event.getX() < 0 || event.getX() > view.getWidth() ||
            event.getY() < 0 || event.getY() > view.getHeight()) {
            result.append(">>> Touch has left the view <<<\n");
        }
        result.append("Edge flags: ").append(
                    event.getEdgeFlags()).append("\n");
        result.append("Pressure: ").append(event.getPressure()).append(" ");
        result.append("Size: ").append(event.getSize()).append("\n");
        result.append("Down time: ").append(
                    event.getDownTime()).append("ms\n");
        result.append("Event time: ").append(
                    event.getEventTime()).append("ms");
        result.append(" Elapsed: ").append(
                    event.getEventTime()-event.getDownTime());
        result.append(" ms\n");
        return result.toString();
    }
}
```

В коде основного явления задаются обратные вызовы, ассоциированные с кнопками, и шаблоны, поэтому мы можем обрабатывать события касания (то есть объекты `MotionEvent`), происходящие с любым элементом нашего пользовательского интерфейса. Мы добавили развернутый механизм журналирования, поэтому

можем точно судить о том, что происходит при событиях касания. Скомпилировав и запустив эту программу, мы увидим экран, показанный на рис. 16.1.

Чтобы максимально эффективно задействовать программу, нужно открыть в Eclipse LogCat и следить за сообщениями, которые будут выводиться, когда вы касаетесь экрана устройства. Этот механизм работает как в эмуляторе, так и в реальном устройстве. Кроме того, рекомендуем максимально расширить окно LogCat, чтобы вы могли прокручивать его при помощи мыши и просмотреть все события, генерируемые этой программой. Чтобы максимально расширить окно LogCat, просто сделайте двойной щелчок на вкладке **LogCat**. Теперь перейдите в пользовательский интерфейс программы, нажмите и отпустите самую верхнюю кнопку, на которой написано **returns true** (возвращает true). Если вы работаете в эмуляторе, сделайте это при помощи мыши. В LogCat после этого должны зарегистрироваться как минимум два события. Сообщения будут помечены как пришедшие от trueBtnTop и будут регистрироваться как относящиеся к методу onTouch() явления MainActivity. В файле MainActivity.java приведен код метода onTouch(). Просматривая LogCat, обратите внимание, какие вызовы методов генерируют значения. Например, значение, отображаемое после Action получено от метода getAction(). В листинге приведен пример журнала LogCat из эмулятора, а в листинге 16.5 — тот же пример из реального устройства.

Листинг 16.4. Пример сообщений LogCat программы TouchDemo1 в эмуляторе

```
trueBtnTop          ----------------------------
trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 0
trueBtnTop          Location: 52.0 x 20.0
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.0 Size: 0.0
trueBtnTop          Down time: 163669ms
trueBtnTop          Event time: 163669ms Elapsed: 0 ms
trueBtnTop          and I'm returning true
trueBtnTop          ----------------------------
trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 1
trueBtnTop          Location: 52.0 x 20.0
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.0 Size: 0.0
trueBtnTop          Down time: 163669ms
trueBtnTop          Event time: 163831ms Elapsed: 162 ms
trueBtnTop          and I'm returning true
```

Листинг 16.5. Пример сообщений LogCat программы TouchDemo1 в реальном устройстве

```
trueBtnTop          ----------------------------
trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 0
trueBtnTop          Location: 42.8374 x 25.293747
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.05490196 Size: 0.2
trueBtnTop          Down time: 24959412ms
```

```
trueBtnTop          Event time: 24959412ms Elapsed: 0 ms
trueBtnTop          and I'm returning true
trueBtnTop          ----------------------------
trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 2
trueBtnTop          Location: 42.8374 x 25.293747
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.05490196 Size: 0.2
trueBtnTop          Down time: 24959412ms
trueBtnTop          Event time: 24959530ms Elapsed: 118 ms
trueBtnTop          and I'm returning true
trueBtnTop          ----------------------------
trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 1
trueBtnTop          Location: 42.8374 x 25.293747
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.05490196 Size: 0.2
trueBtnTop          Down time: 24959412ms
trueBtnTop          Event time: 24959567ms Elapsed: 155 ms
trueBtnTop          and I'm returning true
```

Первое событие имеет значение действия 0, и это действие — ACTION_DOWN. Последнее событие имеет значение действия 1, и это действие — ACTION_UP. В реальном устройстве вам могут встретиться и другие события. Скорее всего, любые события между ACTION_DOWN и ACTION_UP будут иметь значение 2, что соответствует ACTION_MOVE. Еще могут иметь место действие 3 — ACTION_CANCEL и 4 — ACTION_OUTSIDE. Работая пальцами с реальным сенсорным экраном, почти невозможно просто коснуться поверхности и сразу убрать палец, поэтому будьте готовы к незапланированным событиям ACTION_MOVE.

Список различий между эмулятором и реальным устройством этим не ограничивается. Обратите внимание на то, что точность определения положения в эмуляторе ограничена целыми числами (например, 52 на 20), а в реальном устройстве применяются и дроби (42.8374 на 25.293747). При определении места, в котором произошло MotionEvent, выделяются компоненты X и Y, где X соответствует расстоянию от левого края объекта View, в котором произошло касание, а Y — расстоянию от его же верхнего края.

Кроме того, необходимо учитывать, что сила нажатия при работе с эмулятором равна 0, как и размер. При работе с реальным устройством показатель силы нажатия свидетельствует о том, насколько сильным было нажатие на экран и каков был размер площади, накрытой пальцем. Если легко касаться экрана кончиком мизинца, значения сила нажатия и площади будут невелики. Если же сильно нажать большим пальцем, оба этих показателя возрастут. В документации сказано, что значения силы нажатия и размера должны варьироваться между 0 и 1. Однако из-за аппаратных различий устройств в программе может быть очень сложно обойтись какими-либо абсолютными значениями, если речь идет о силе нажатия и размере. Было бы целесообразно сравнивать эти значения между разными MotionEvent, происходящими в вашей программе. Но если вы определите, например, что показатель силы нажатия выше 0.8 свидетельствует о слишком сильном нажатии, могут на-

чаться проблемы. На конкретном устройстве вы можете никогда не получить не то что 0.8, но и 0.2.

ПРИМЕЧАНИЕ

В листинге 16.5 видно, что в ходе последовательности касаний при работе с реальным устройством показатели силы нажатия не изменяются. Это обусловлено ошибкой в коде Android для устройства Motorolla Droid (Android 2.0), которая вскоре будет исправлена. Как правило, по ходу последовательности касаний на реальном устройстве показатели силы нажатия и размера должны изменяться по мере перехода от ACTION_DOWN к ACTION_MOVE и, наконец, к ACTION_UP.

Значения времени простоя (down time) и времени события (event time) одинаково обрабатываются как в эмуляторе, так и в реальном устройстве, вся разница в том, что значения на реальном устройстве гораздо выше. Значения истекшего времени (elapsed time) обрабатываются одинаково.

Флаги граней (edge flags) предназначены для того, чтобы определять, когда касание произойдет у края физического (реального) экрана. В документации Android SDK говорится, что такие флаги играют роль индикаторов, указывающих, что точка касания стыкуется с краем (правым, левым, верхним или нижним). Однако метод getEdgeFlags() может всякий раз возвращать 0, в зависимости от того, с каким эмулятором или устройством вы работаете. На некотором оборудовании очень сложно определить, на самом ли деле касание произошло у края экрана, поэтому Android «прикрепляет» определенное местоположение к грани и устанавливает нужный флаг за вас. Это происходит не всегда, поэтому рекомендуется дополнительно проверять правильность расстановки флагов граней. В классе MotionEvent имеется метод setEdgeFlags(), поэтому, если хотите, можете расставлять флаги граней самостоятельно.

Здесь также необходимо отметить, что метод onTouch() возвращает true, поскольку элемент TrueButton запрограммирован на возврат true. Android интерпретирует возврат true как знак того, что объект MotionEvent обработан, поэтому нет смысла передавать его еще какому-то элементу. Кроме того, Android продолжает посылать данные о событиях касания в рамках текущей последовательности этому методу. При работе с реальным устройством несложно понять, как мы получаем события ACTION_UP и ACTION_MOVE.

Теперь коснемся пальцем кнопки **return false** (возвращает false) в верхней части экрана. В оставшейся части главы мы будем приводить примеры вывода LogCat только с устройства, но не с эмулятора. Разница уже объяснена, поэтому, работая с эмулятором, вы должны понимать, почему видите именно такие результаты. В листинге 16.6 показан образец вывода LogCat при касании кнопки **returns false** (возвращает false).

Листинг 16.6. Пример LogCat при касании верхней кнопки returns false

```
falseBtnTop        -----------------------------
falseBtnTop        Got view falseBtnTop in onTouch
falseBtnTop        Action: 0
falseBtnTop        Location: 61.309372 x 44.281494
falseBtnTop        Edge flags: 0
falseBtnTop        Pressure: 0.0627451 Size: 0.26666668
falseBtnTop        Downtime: 28612178ms
```

```
falseBtnTop        Event time: 28612178ms Elapsed: 0 ms
falseBtnTop        and I'm returning false
falseBtnTop        --------------------------------
falseBtnTop        Action: 0
falseBtnTop        Location: 61.309372 x 44.281494
falseBtnTop        Edge flags: 0
falseBtnTop        Pressure: 0.0627451 Size: 0.26666668
falseBtnTop        Downtime: 28612178ms
falseBtnTop        Event time: 28612178ms Elapsed: 0 ms
falseBtnTop        super onTouchEvent() returns true
falseBtnTop        and I'm returning false
trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 0
trueLayoutTop        Location: 61.309372 x 116.281494
trueLayoutTop        Edge flags: 0
trueLayoutTop        Pressure: 0.0627451 Size: 0.26666668
trueLayoutTop        Downtime: 28612178ms
trueLayoutTop        Event time: 28612178ms Elapsed: 0 ms
trueLayoutTop        and I'm returning true
trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 2
trueLayoutTop        Location: 61.309372 x 111.90039
trueLayoutTop        Edge flags: 0
trueLayoutTop        Pressure: 0.0627451 Size: 0.26666668
trueLayoutTop        Downtime: 28612178ms
trueLayoutTop        Event time: 28612217ms Elapsed: 39 ms
trueLayoutTop        and I'm returning true
trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 1
trueLayoutTop        Location: 55.08958 x 115.30792
trueLayoutTop        Edge flags: 0
trueLayoutTop        Pressure: 0.0627451 Size: 0.26666668
trueLayoutTop        Downtime: 28612178ms
trueLayoutTop        Event time: 28612361ms Elapsed: 183 ms
trueLayoutTop        and I'm returning true
```

Теперь ситуация значительно изменилась, разберемся, что же произошло. Android получает событие ACTION_DOWN объекта MotionEvent и передает его нашему методу onTouch() из класса MainActivity. Метод onTouch() записывает данные в LogCat и возвращает false. Так мы сообщаем Android, что метод onTouch() не обработал событие, и Android ищет следующий метод, который следует вызвать. В нашем случае такой метод переопределяется методом onTouchEvent() из класса FalseButton. Поскольку класс FalseButton является дополнением класса BooleanButton, обратитесь к методу onTouchEvent() в классе BooleanButton.java и изучите код. В методе onTouchEvent() мы вновь записываем информацию в LogCat, вызываем метод onTouchEvent() родительского класса и в ответ снова получаем false. Обратите внимание — информация о местоположении в LogCat так и не изменилась. Этого

следовало ожидать, так как мы все еще работаем с тем же объектом View, что и в случае с FalseButton. Мы видим, что наш родительский класс собирается возвратить true от onTouchEvent(), и можем понять почему. Если сейчас взглянуть на кнопку **returns false** (возвращает false) в пользовательском интерфейсе, она будет отличаться от **returns true** (возвращает true) цветом. Теперь кнопка **returns false** (возвращает false) выглядит наполовину утопленной. То есть создается впечатление, что ее нажали, но еще не отпустили. Наш пользовательский метод вернул false, а не true. Поскольку мы вновь сообщили Android, что это событие не было обработано (так как вернули false), Android никогда не отправит нашей кнопке событие ACTION_UP, следовательно, она так и «не узнает», что палец поднялся с экрана. Поэтому наша кнопка по-прежнему нажата. Если бы мы вернули true, как и требовал наш родительский класс, то в итоге получили бы событие ACTION_UP и могли бы вновь изменить цвет кнопки на стандартный. Резюмируем: всякий раз, возвращая false от объекта пользовательского интерфейса в ответ на получение MotionEvent, Android прекращает посылать объекты MotionEvent данному элементу пользовательского интерфейса и ищет в пользовательском интерфейсе другой объект, который мог бы обработать MotionEvent.

Вероятно, вы заметили, что при прикосновении к кнопке **returns true** (возвращает true) ее цвет не изменился. Почему? Это происходит потому, что метод onTouch() вызывался ранее, чем все методы кнопок, и потому, что onTouch() возвратил true. Android никогда не «утруждает» себя вызовом метода onTouchEvent() кнопки **returns true** (возвращает true). Если добавить к методу onTouch() строку v.onTouchEvent(event); перед самым моментом возврата true, кнопка будет менять цвет. Кроме того, новые строки добавятся и в LogCat, так как метод onTouchEvent() также будет записывать информацию в LogCat.

Вернемся к изучению вывода LogCat. Теперь, когда Android уже дважды пытался найти обработчик для события ACTION_DOWN и ему это не удалось, он переходит к следующему объекту View данной программы, который, возможно, сможет получить событие. В нашем случае таким видом является шаблон под кнопкой. Мы вызываем верхний шаблон trueLayoutTop и видим, что он получил событие ACTION_DOWN.

Обратите внимание: был еще один вызов onTouch(), но на этот раз не с видом кнопки, а с видом шаблона. Почти все данные объекта MotionEvent, переданные onTouch() для trueLayoutTop, те же, что и ранее, в том числе показатели времени. Отличается лишь координата местоположения Y. Координата Y для кнопки была равна 44.281494, а для шаблона — 116.281494. И это логично, так как кнопка находится не в верхнем левом углу шаблона, а под кнопкой **returns true** (возвращает true). Поэтому координата Y точки касания относительно шаблона больше, чем координата Y того же касания относительно кнопки. Касание произошло дальше от верхнего края шаблона, чем от верхнего края кнопки. Поскольку метод onTouch(), относящийся к trueLayoutTop, возвращает true, Android посылает все оставшиеся события касания шаблону и в журнале появляются записи, соответствующие событиям ACTION_MOVE и ACTION_UP.

Вновь коснемся верхней кнопки **return false** (возвращает false) и обратим внимание на то, что в журнале появляется тот же набор записей. То есть onTouch() вы-

зывается для falseBtnTop; onTouchEvent() вызывается для falseBtnTop, затем onTouch() вызывается для trueLayoutTop и для оставшихся событий. Android прекращает посылать кнопке только те события, которые связаны с отдельно взятой последовательностью касаний. Если возникнет новая последовательность касаний, то Android будет посылать события кнопке, пока вновь не получит false от вызванного метода, который все еще выполняется в нашей программе-примере.

Теперь прикоснитесь к верхнему шаблону, но не трогайте кнопки, потом немного проведите пальцем от экрана — и уберите палец. (Если вы работаете с эмулятором, постарайтесь воспроизвести те же движения при помощи мыши.) Обратите внимание на поток записей в LogCat, где первая из них соответствует событию ACTION_DOWN, затем идет множество событий ACTION_MOVE и, наконец, ACTION_UP.

А сейчас прикоснитесь к верхней кнопке **returns true** (возвращает true), но прежде, чем убрать палец, проведите им по экрану. В листинге 16.7 показана новая информация, появившаяся в LogCat.

Листинг 16.7. Записи LogCat, описывающие касание за пределами вида

```
[ … log messages of an ACTION_DOWN event followed by some
                                    ACTION_MOVE events … ]

trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 2
trueBtnTop          Location: 150.41768 x 22.628128
trueBtnTop          >>> Touch has left the view <<<
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.047058824 Size: 0.13333334
trueBtnTop          Downtime: 31690859ms
trueBtnTop          Event time: 31691344ms Elapsed: 485 ms
trueBtnTop          and I'm returning true

[ … more ACTION_MOVE events logged … ]

trueBtnTop          Got view trueBtnTop in onTouch
trueBtnTop          Action: 1
trueBtnTop          Location: 291.5864 x 223.43854
trueBtnTop          >>> Touch has left the view <<<
trueBtnTop          Edge flags: 0
trueBtnTop          Pressure: 0.047058824 Size: 0.13333334
trueBtnTop          Downtime: 31690859ms
trueBtnTop          Event time: 31692493ms Elapsed: 1634 ms
trueBtnTop          and I'm returning true
```

Даже после того, как палец будет убран с кнопки, мы продолжим получать уведомления о прикосновении к кнопке. Первая запись в листинге 16.7 относится к событию, которое происходит, когда мы уже не трогаем кнопку. В таком случае координата X события касания находится правее края нашего объекта-кнопки. Но мы продолжаем получать вызовы от объектов MotionEvent, пока не произойдет событие ACTION_UP. Это объясняется тем, что метод onTouch() продолжает возвращать true. Даже когда мы наконец оторвем палец от сенсорного экрана и палец не будет

касаться кнопки, продолжаются вызовы метода onTouch(), ожидающего события ACTION_UP, а пока возвращающего true. Об этом необходимо помнить, работая с MotionEvent. Когда палец сдвинется с элемента (вида), мы можем прекратить любую операцию, выполнявшуюся в то время, пока палец был на виде, и вернуть false от метода onTouch(), чтобы не получать уведомления о дальнейших событиях. Либо мы можем продолжить получать события (вернув true от метода onTouch()) и выполнять логику только в том случае, если до отрыва от экрана палец вновь окажется на виде.

Последовательность событий касания оказалась связана с верхней кнопкой **returns true** (возвращает true) тогда, когда мы вернули true от метода onTouch(). Так мы сообщили Android, что можно прекратить поиск объекта, который мог бы принять объекты MotionEvent, и приказали просто присылать нам все дальнейшие объекты MotionEvent, относящиеся к данной последовательности касаний. Даже если, проводя пальцем по экрану, мы затронем другой вид, эта последовательность по-прежнему останется привязана к первому виду.

Рассмотрим, что происходит в нижней части нашего приложения. Для этого коснемся кнопки **returns true** (возвращает true), расположенной в нижней части экрана. Ситуация будет та же, что и с верхней кнопкой **returns true** (возвращает true). Поскольку onTouch() возвращает true, Android посылает нам оставшиеся события последовательности касаний, пока палец не оторвется от сенсорного экрана. Далее коснемся нижней кнопки **return false** (возвращает false). Опять же метод onTouch() возвращает false, и метод onTouchEvent() также возвращает false (они оба связаны с объектом falseBtnBottom). В этой ситуации следующим видом, который будет получать события MotionEvent, является объект falseLayoutBottom, также возвращающий false. Итак, готово. Поскольку метод onTouchEvent() вызвал вышестоящий метод onTouchEvent(), кнопка изменяет цвет, чтобы указать, что она наполовину утоплена. Но, как и в более раннем случае, кнопка так и останется нажатой, поскольку в текущей последовательности касаний никогда не произойдет события ACTION_UP — ведь наши методы все время возвращают false. В отличие от предыдущих случаев, это событие не относится даже к шаблону. Если прикоснуться к нижней кнопке **returns false** (возвращает false) и удерживать ее, а затем провести пальцем по экрану, в LogCat не появится никаких новых записей, поскольку к нам больше не будет прислан ни один объект MotionEvent. Мы все время возвращаем false, поэтому Android и не беспокоит нас больше никакими событиями, относящимися к данной последовательности касаний. Но если мы начнем новую последовательность касаний, то увидим, как в LogCat появляются новые записи. Если начать последовательность касаний в нижнем шаблоне (но не с кнопки), то в LogCat отобразится единственное событие — для falseLayoutBottom, кнопка вернет false, и на этом все закончится (пока мы не начнем новую последовательность касаний).

До сих пор мы использовали кнопки для того, чтобы продемонстрировать эффект событий MotionEvent на сенсорном экране. Следует отметить, что, как правило, логика для работы с кнопками реализуется при помощи метода onClick(). В этой программе-примере мы использовали кнопки потому, что их просто создавать и они являются субклассами View, то есть могут получать события точно так же, как любой другой вид. Не забывайте, что эти техники применяются к любому объекту View нашей программы, независимо от того, является ли класс View стандартным или пользовательским.

Работа с VelocityTracker

В Android есть класс, используемый для обработки последовательностей касаний при работе с сенсорным экраном. Он называется VelocityTracker. Когда палец движется по сенсорному экрану, может быть полезно знать его скорость. Например, если пользователь двигает палец быстро, это может требовать применения в программе специальной логики. Класс Android VelocityTracker помогает справиться с сопутствующими математическими вычислениями.

Для работы с VelocityTracker сначала нужно получить экземпляр VelocityTracker. Для этого вызывается статический метод VelocityTracker.obtain(). Затем к нему можно добавить объекты MotionEvent при помощи метода addMovement(MotionEvent ev). Этот метод нужно вызывать с обработчиком, получающим объекты MotionEvent, — таков, например, метод обработчика onTouch() или метод вида onTouchEvent().

VelocityTracker использует объекты MotionEvent, чтобы определить, что именно происходит с пользовательской последовательностью касаний. Когда в VelocityTracker оказывается как минимум два объекта MotionEvent, при помощи других методов мы можем узнать, что именно происходит.

Два метода класса VelocityTracker — getXVelocity() и getYVelocity() — возвращают соответствующую скорость пальца в направлениях X и Y. Значения, возвращаемые двумя этими методами, представляют собой количество пикселов за единицу времени. Это могут быть пикселы в миллисекунду, пикселы в секунду или другие подобные единицы на ваше усмотрение. Чтобы сообщить VelocityTracker, какой период времени будет использоваться, перед вызовом двух этих методов-получателей (getter method) нужно активировать метод computeCurrentVelocity(int units) класса VelocityTracker. Значение units показывает, в течение скольких миллисекунд измеряется скорость. Если вы хотите работать с пикселами в миллисекунду, используйте единичные значения, если с пикселами в секунду — значения, кратные 1000. Значения, возвращаемые методами getXVelocity() и getYVelocity(), будут положительными, если движение направлено вправо (для X) или вниз (для Y). Соответственно при движении влево (для X) или вверх (для Y) имеем отрицательные значения.

Когда закончите работу с объектом VelocityTracker, полученным при помощи метода obtain(), вызовите метод recycle() объекта VelocityTracker. В листинге 16.8 показан пример обработчика onTouchEvent() для работы с видом.

Листинг 16.8. Пример обработчика, использующего VelocityTracker

```
private VelocityTracker vTracker = null;

@Override
public boolean onTouchEvent(MotionEvent event) {
   int action = event.getAction();
   switch(action) {
      case MotionEvent.ACTION_DOWN:
         if(vTracker == null) {
            vTracker = VelocityTracker.obtain();
         }
         else {
            vTracker.clear();
```

```
            }
            vTracker.addMovement(event);
            break;
        case MotionEvent.ACTION_MOVE:
            vTracker.addMovement(event);
            vTracker.computeCurrentVelocity(1000);
            Log.v(TAG, "X velocity is " + vTracker.getXVelocity() +
                " pixels per second");
            Log.v(TAG, "Y velocity is " + vTracker.getYVelocity() +
                " pixels per second");
            break;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            vTracker.recycle();
            break;
    }
    event.recycle();
    return true;
}
```

Еще несколько слов о VelocityTracker. Очевидно, что, если добавить к VelocityTracker лишь один MotionEvent (например, событие ACTION_DOWN), скорость может быть только нулевой. Вместо этого нужно задать начальную точку, за которой последуют события ACTION_MOVE, и, опираясь на них, можно вычислить скорость. Оказывается, что скорости, вычисляемые после добавления к VelocityTracker после ACTION_UP, также равны нулю. Поэтому, не пытайтесь вычислить движение по значениям X и Y, следующим за ACTION_UP. Если вы пишете игру, в которой пользователь должен бросать объект, расположенный на экране, начинайте работать со скоростями после добавления последнего события ACTION_MOVE, чтобы получить траекторию, которую имеет объект в игровом поле. VelocityTracker тратит на работу немало ресурсов, поэтому используйте его осторожно. Кроме того, не забывайте повторно применять этот класс, как только закончите выполнять с его помощью определенную задачу. В Android могут одновременно использоваться несколько экземпляров VelocityTracker, и при этом они могут тратить немало памяти. По этой причине завершайте их, как только прекращаете с ними работать. В листинге 16.8 также используется метод clear() — в случае если начинается новая последовательность касаний (то есть мы получаем событие ACTION_DOWN, а наш объект VelocityTracker уже существует). Он позволяет обойтись без повторного использования имеющегося объекта и получить новый.

Изучение функции перетаскивания (Drag and Drop)

Итак, вы уже умеете получать объекты MotionEvent в коде. Теперь научимся делать с ними кое-что интересное. Мы поговорим о внедрении в программу функции перетаскивания (Drag and Drop). Для начала «потаскаем» что-нибудь. В следующей демонстрационной программе мы возьмем белую точку и перенесем ее в другое место в нашем шаблоне. Опираясь на листинг 16.9, сделаем новый проект Android и создадим по указанному образцу XML-файл, а в код Java добавим новый класс

под названием Dot. Обратите внимание: имя пакета в XML-шаблоне для элемента Dot должно совпадать с именем пакета, применяемого для всей программы. Кроме того, отметим, что главный класс Activity можно оставить без изменений — он и так неплох. Пользовательский интерфейс этой программы показан на рис. 16.2.

Листинг 16.9. XML-шаблон и код Java для программы с функцией перетаскивания

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     >

  <com.androidbook.touch.dragdemo1.Dot
     android:id="@+id/dot"
     android:tag="trueDot"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content" />

</LinearLayout>

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.MotionEvent;
import android.view.View;

public class Dot extends View {
 private static final float RADIUS = 20;
 private float x = 30;
 private float y = 30;
 private float initialX;
 private float initialY;
 private float offsetX;
 private float offsetY;
 private Paint backgroundPaint;
 private Paint myPaint;

 public Dot(Context context, AttributeSet attrs) {
     super(context, attrs);

     backgroundPaint = new Paint();
     backgroundPaint.setColor(Color.BLUE);

     myPaint = new Paint();
     myPaint.setColor(Color.WHITE);
```

```
        myPaint.setAntiAlias(true);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        int action = event.getAction();
        switch(action) {
        case MotionEvent.ACTION_DOWN:
            // необходимо запомнить первичную стартовую точку
            // центр - это Dot, откуда начинается последовательность касаний
            initialX = x;
            initialY = y;
            offsetX = event.getX();
            offsetY = event.getY();
            break;
        case MotionEvent.ACTION_MOVE:
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL:
            x = initialX + event.getX() - offsetX;
            y = initialY + event.getY() - offsetY;
            break;
        }
        event.recycle();
        return(true);
    }

    @Override
    public void draw(Canvas canvas) {
        int width = canvas.getWidth();
        int height = canvas.getHeight();
        canvas.drawRect(0, 0, width, height, backgroundPaint);

        canvas.drawCircle(x, y, RADIUS, myPaint);
        invalidate();
    }
}
```

Запустив эту программу, мы видим белую точку на синем фоне. Можете тронуть точку пальцем и перетащить ее в другое место на экране. Когда вы отпускаете палец, точка остается на месте, пока вы снова не прикоснетесь к ней и не перетащите куда-нибудь еще. Мы предельно упростили этот пример, чтобы продемонстрировать в общих чертах, как передвигать объект на экране. Метод draw() помещает точку в ее актуальном месте с координатами X и Y. Получая объекты MotionEvent в методе onTouchEvent(), можно изменять значения X и Y, касаясь экрана в другом месте. Начальная позиция точки записывается в методе ACTION_DOWN, там же указывается и первичная точка прикосновения. Поскольку мы не всегда касаемся объекта прямо в его центре, координаты прикосновения и координаты объекта не будут совпадать. Кроме того, если точкой привязки объекта служит не его центр, а, например, верхний левый угол, нужно учитывать и это обстоятельство. Когда палец начинает двигаться по экрану, мы корректируем положение объекта, ориентируясь на начальные и конечные точки X и Y (до начала перемещения и после перемещения)

на основании получаемых событий MotionEvent. Когда движение прекращается (то есть происходит ACTION_UP), мы окончательно определяем положение, используя координаты последнего касания. Здесь мы немного жульничаем, поскольку наш вид Dot расположен на экране относительно начала координат (0,0). Это значит, что мы можем просто нарисовать круг относительно (0,0), противопоставив его другой точке отсчета. Если объект размещен не относительно (0,0), а как-то иначе, то нам может понадобиться предоставить дополнительные отступы, чтобы установить положение нашего объекта. В этом примере мы, кроме того, обходимся без полос прокрутки. Они могут усложнить задачу вычисления положения нашего объекта на экране. Но с полосами прокрутки базовые принципы работы не изменятся. Зная стартовое положение объекта, который предполагается передвинуть, и отслеживая точки, в которых располагается объект до начала перемещения и после перемещения (delta values) наших касаний между событиями ACTION_DOWN и ACTION_UP, мы можем корректировать положение объекта на экране.

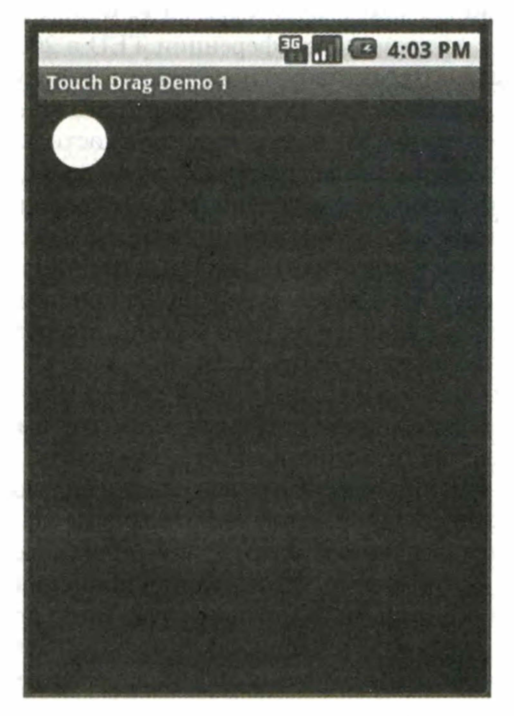


Рис. 16.2. Пользовательский интерфейс программы, демонстрирующей функцию перетаскивания

Чтобы вложить один экранный объект в другой, важно знать, где именно объекты расположены на экране, работа с самими касаниями уже второстепенна. Мы не будем приводить здесь пример такого вложения, но объясним процесс в общих чертах. Ранее было показано, что при перетаскивании объекта по экрану мы знаем его положение относительно одной или более точек привязки. Кроме того, можно запрашивать у экранных объектов данные об их положении и размере. На основании этой информации можно определить, оказался ли перемещенный

объект «поверх» другого. Обычно процесс определения целевого пункта перемещения (drop target) для перетаскиваемого объекта строится так: осуществляется итерация всех объектов, на которые он может быть наложен, и определяется, не пересекается ли его актуальная позиция с одним из этих объектов. Чтобы это узнать, можно пользоваться размерами и положением (а иногда — очертаниями) каждого из этих объектов. Если происходит событие ACTION_UP (значит, что пользователь отпустил перетаскиваемый объект, а сам объект в результате оказался на что-то наложен), то мы можем запустить логику, обрабатывающую акт наложения. В результате возможны следующие варианты: какой-то объект отправляется в корзину либо файл копируется или переносится в определенную папку.

Технология мультитач

Теперь, когда мы умеем работать с одиночными касаниями, изучим и множественные. Для работы с ними предназначена технология мультитач. Мультитач вызывает огромный интерес со времени конференции TED в 2006 году, когда Джефф Хан продемонстрировал интерактивную поверхность для компьютерных пользовательских интерфейсов. Если экран может реагировать на движения по нему нескольких пальцев, это открывает широкие возможности для управления экранными объектами. Например, если дотронуться двумя пальцами до изображения, а потом развести их, это изображение увеличится. Расположив несколько пальцев на изображении и повернув их по часовой стрелке, вы можете соответствующим образом изменить положение картинки на экране. В Android технология мультитач поддерживается в версиях 2.0 и выше. В версии 2.0 при помощи трех пальцев на экране можно выполнять увеличение, уменьшение, вращение и многое другое. Во всем этом нет ни капли волшебства. Если материал, из которого изготовлен экран, может улавливать несколько прикосновений по мере того, как они происходят на экране, может уведомлять программу о том, как именно прикосновения перемещаются по поверхности экрана, и в итоге определять, когда прикосновения к экрану прекратятся, то программа может понять, что пользователь хочет сделать при помощи этих движений. Пусть это и не волшебство, но достаточно сложный механизм. В этом разделе мы поможем вам лучше разобраться с мультитач.

Основы мультитач в целом те же, что и у одиночного прикосновения. Для прикосновений создаются объекты MotionEvent, которые затем передаются методам — все как и раньше. Код считывает информацию о прикосновениях и программа решает, что делать дальше. На базовом уровне MotionEvent работает с теми же методами, что и при одиночном касании, — то есть мы вызываем getAction(), getDownTime(), getX() и т. д. Однако, если мы касаемся экрана несколькими пальцами, в MotionEvent должна учитываться информация о каждом из них, а здесь уже есть несколько оговорок. От метода getAction() мы получаем значение для одного пальца, а не для всех. Значение момента прикосновения (down time) учитывается только для первого прикосновения к экрану и сохраняется все тем же, пока на экране остается хотя бы один палец. Значения местоположения getX() и getY(), а также getPressure() и getSize() могут принимать аргумент для одного пальца, то есть нам требуется особое индексное значение для запроса информации именно о том пальце, положение которого нас интересует. Некоторые из рассмотренных выше вызовов мето-

дов не принимают никаких аргументов для указания пальца (таковы, например, getX(), getY()), поэтому, если мы будем работать с ними, то как мы поймем, к какому именно пальцу будут относиться содержащиеся в них значения? Это можно определить, но не без труда. Следовательно, если не следить все время за несколькими пальцами, на выходе можно получить весьма странные результаты. Но давайте, наконец, разберемся, что нужно делать.

Первый метод MotionEvent, с которым нужно познакомиться при работе с мультитач — это getPointerCount(). Он сообщает, сколько именно пальцев задействовано в работе с объектом MotionEvent. Этот метод не позволяет точно узнать, сколько пальцев касается экрана в данный момент, поскольку это уже зависит от оборудования и от Android. Может оказаться, что в каких-то устройствах getPointerCount() сообщает не обо всех пальцах, прикасающихся к экрану, а только о некоторых. Но давайте продолжим. Как только получена информация о том, что с объектом MotionEvent работают несколько пальцев, нужно задействовать индекс указателей (pointer index) и ID указателей.

Объект MotionEvent содержит информацию об указателях, начиная с номера 0 и до номера, равного количеству пальцев, работающих с этим объектом. Индекс указателей всегда начинается с 0. Если на объекте три пальца, то в индексе будет три номера: 0, 1 и 2. При вызове методов, например getX(), нужно указывать номер пальца, информацию о котором вы хотите получить. ID указателей являются натуральными числами, показывающими, какой палец отслеживается в данный момент. ID указателей начинаются с 0, но такая ситуация сохраняется не всегда, так как с экрана постоянно убираются одни пальцы и появляются новые. ID указателя можно сравнить с именем пальца, присваиваемым на время отслеживания пальца в Android. Например, представим себе две последовательности касаний в следующем порядке: опущен первый палец, опущен второй палец, поднят первый палец, поднят второй палец. Первый опущенный палец получит ID указателя 0. Второй опущенный палец получит ID указателя 1. Как только первый палец будет поднят с экрана, второй палец все еще будет иметь ID 1. Но в таком случае индексное значение для второго пальца станет равно 0, так как индекс всегда должен начинаться с 0. В данном примере второй палец начинает работу с индексным значением 1, но получает значение 0, как только первый палец оказывается убран с экрана. Программы будут использовать ID указателей для ассоциирования событий, связанных с конкретным пальцем, а также всех других пальцев, участвующих в работе. Рассмотрим пример.

В листинге 16.10 показан новый XML-шаблон, а также код Java для программы с технологией мультитач. При помощи листинга 16.10 создаем новое приложение, а затем запускаем его. На рис. 16.3 показано, как оно должно выглядеть.

Листинг 16.10. XML-шаблон и код Java для программы, использующей мультитач

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:id="@+id/layout1"
 android:tag="trueLayout"
 android:orientation="vertical"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
```

```
 android:layout_weight="1"
 >

 <TextView android:text="Touch fingers on the screen and look at LogCat"
 android:id="@+id/message"
 android:tag="trueText"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_alignParentBottom="true" />

</RelativeLayout>

// это файл MainActivity.java
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.RelativeLayout;

public class MainActivity extends Activity implements OnTouchListener {
 /** Вызывается при первом создании явления. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     RelativeLayout layout1 = (RelativeLayout)
             findViewById(R.id.layout1);
     layout1.setOnTouchListener(this);
 }

 @Override
 public boolean onTouch(View v, MotionEvent event) {
     String myTag = v.getTag().toString();
     Log.v(myTag, "----------------------------");
     Log.v(myTag, "Got view " + myTag + " in onTouch");
     Log.v(myTag, describeEvent(event));
     if( "true".equals(myTag.substring(0, 4))) {
         Log.v(myTag, "and I'm returning true");
         return true;
     }
     else {
         Log.v(myTag, "and I'm returning false");
         return false;
     }
 }

 protected static String describeEvent(MotionEvent event) {
     StringBuilder result = new StringBuilder(500);
     result.append("Action: ").append(event.getAction()).append("\n");
```



Рис. 16.3. Наше демонстрационное приложение с технологией мультитач

```
int numPointers = event.getPointerCount();
    result.append("Number of pointers: ").append(
                numPointers).append("\n");
    int ptrIdx = 0;
    while (ptrIdx < numPointers) {
        int ptrId = event.getPointerId(ptrIdx);
        result.append("Pointer Index: ").append(ptrIdx);
        result.append(", Pointer Id: ").append(ptrId).append("\n");
        result.append(" Location: ").append(event.getX(ptrIdx));
        result.append(" x ").append(event.getY(ptrIdx)).append("\n");
        result.append(" Pressure: ").append(event.getPressure(ptrIdx));
        result.append(" Size: ").append(
                event.getSize(ptrIdx)).append("\n");

        ptrIdx++;
    }
    result.append("Downtime: ").append(
                event.getDownTime()).append("ms\n");
    result.append("Event time: ").append(
                event.getEventTime()).append("ms");
    result.append(" Elapsed: ").append(
                event.getEventTime()-event.getDownTime());
    result.append(" ms\n");
    return result.toString();
 }
}
```

Если мы просто испытываем эту программу в эмуляторе, она все равно будет работать, но мы не сможем одновременно работать с несколькими пальцами на экране. Вывод будет примерно таким же, как и с предыдущей программой. В листинге 16.11 приведены записи из LogCat для рассмотренной выше последовательности касаний, то есть первый палец касается экрана, второй палец касается экрана, первый палец отрывается от экрана, второй палец отрывается от экрана.

Листинг 16.11. Образец вывода LogCat для программы с технологией мультитач

```
trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 0
trueLayoutTop        Number of pointers: 1
trueLayoutTop        Pointer Index: 0, Pointer Id: 0
trueLayoutTop           Location: 722.3844 x 94.37604
trueLayoutTop           Pressure: 0.07450981 Size: 0.2
trueLayoutTop        Downtime: 15778221ms
trueLayoutTop        Event time: 15778221ms Elapsed: 0 ms
trueLayoutTop        and I'm returning true

trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 2
trueLayoutTop        Number of pointers: 1
trueLayoutTop        Pointer Index: 0, Pointer Id: 0
trueLayoutTop           Location: 722.3844 x 97.29675
trueLayoutTop           Pressure: 0.07450981 Size: 0.2
trueLayoutTop        Downtime: 15778221ms
trueLayoutTop        Event time: 15778470ms Elapsed: 249 ms
trueLayoutTop        and I'm returning true
trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 261
trueLayoutTop        Number of pointers: 2
trueLayoutTop        Pointer Index: 0, Pointer Id: 0
trueLayoutTop           Location: 722.3844 x 98.75711
trueLayoutTop           Pressure: 0.07450981 Size: 0.2
trueLayoutTop        Pointer Index: 1, Pointer Id: 1
trueLayoutTop           Location: 343.8656 x 103.625
trueLayoutTop           Pressure: 0.06666667 Size: 0.2
trueLayoutTop        Downtime: 15778221ms
trueLayoutTop        Event time: 15778499ms Elapsed: 278 ms
trueLayoutTop        and I'm returning true

trueLayoutTop        ----------------------------
trueLayoutTop        Got view trueLayoutTop in onTouch
trueLayoutTop        Action: 2
trueLayoutTop        Number of pointers: 2
trueLayoutTop        Pointer Index: 0, Pointer Id: 0
trueLayoutTop           Location: 702.8365 x 100.704285
trueLayoutTop           Pressure: 0.07450981 Size: 0.2
trueLayoutTop        Pointer Index: 1, Pointer Id: 1
```

```
trueLayoutTop          Location: 343.8656 x 95.836395
trueLayoutTop          Pressure: 0.06666667 Size: 0.2
trueLayoutTop       Downtime: 15778221ms
trueLayoutTop       Event time: 15778785ms Elapsed: 564 ms
trueLayoutTop       and I'm returning true
trueLayoutTop       ----------------------------
trueLayoutTop       Got view trueLayoutTop in onTouch
trueLayoutTop       Action: 6
trueLayoutTop       Number of pointers: 2
trueLayoutTop       Pointer Index: 0, Pointer Id: 0
trueLayoutTop          Location: 702.8365 x 100.704285
trueLayoutTop          Pressure: 0.07450981 Size: 0.2
trueLayoutTop       Pointer Index: 1, Pointer Id: 1
trueLayoutTop          Location: 343.8656 x 95.34961
trueLayoutTop          Pressure: 0.06666667 Size: 0.2
trueLayoutTop       Downtime: 15778221ms
trueLayoutTop       Event time: 15778812ms Elapsed: 591 ms
trueLayoutTop       and I'm returning true

trueLayoutTop       ----------------------------
trueLayoutTop       Got view trueLayoutTop in onTouch
trueLayoutTop       Action: 2
trueLayoutTop       Number of pointers: 1
trueLayoutTop       Pointer Index: 0, Pointer Id: 1
trueLayoutTop          Location: 343.8656 x 94.86282
trueLayoutTop          Pressure: 0.07450981 Size: 0.2
trueLayoutTop       Downtime: 15778221ms
trueLayoutTop       Event time: 15778825ms Elapsed: 604 ms
trueLayoutTop       and I'm returning true

trueLayoutTop       ----------------------------
trueLayoutTop       Got view trueLayoutTop in onTouch
trueLayoutTop       Action: 1
trueLayoutTop       Number of pointers: 1
trueLayoutTop       Pointer Index: 0, Pointer Id: 1
trueLayoutTop          Location: 323.42917 x 92.42886
trueLayoutTop          Pressure: 0.07450981 Size: 0.2
trueLayoutTop       Downtime: 15778221ms
trueLayoutTop       Event time: 15779138ms Elapsed: 917 ms
trueLayoutTop       and I'm returning true
```

Теперь рассмотрим, что именно происходит с этой программой. Первое событие — это ACTION_DOWN первого пальца. Мы узнаем о нем при помощи метода getAction(). Еще раз обратимся к методу describeEvent() из файла MainActivity.java, чтобы отследить, какие методы дают какой вывод. Получаем один указатель с индексом 0 и ID 0. После этого мы, вероятно, увидим несколько событий ACTION_MOVE, совершаемых первым пальцем. У нас по-прежнему один указатель, его индекс и ID равны 0. Потом экрана касается второй палец. Теперь действие имеет десятичное значение — 261. Что это значит? Значение действия (action value) состоит из двух частей: индикатора, обозначающего, какой из указателей совершил действие, и указания того, что это было за действие. Преобразуем десятичное значение 261

в шестнадцатеричную систему и получаем 0x00000105. Действие — это наименьший байт (в данном случае — 5), а ID указателя — 1. Обратите внимание, что таким способом мы узнаем именно ID, а не индекс указателя. Если бы мы коснулись экрана третьим пальцем, действие было бы равно 0x00000205 (в десятичной системе — 517). При появлении четвертого пальца получилось бы 0x00000305 (773 в десятичном выражении) и т. д.

Рассмотрим две следующие записи из LogCat в листинге 16.11. Первая запись — это событие ACTION_MOVE. Помните, что при работе с реальным экраном очень сложно не совершать пальцами случайных движений. Когда мы поднимаем с экрана первый палец, получаем значение действия, очень похожее на составное значение, возникающее при опускании пальца, но действие равно 6, а не 5. При отрывании от экрана первого пальца в ситуации мультитач мы получаем значение действия 0x00000006 (6 в десятичной системе). Если бы в той же ситуации мы подняли второй палец, значение действия было бы 0x00000106 (262 в десятичной системе). Обратите внимание — у нас по-прежнему есть информация от двух пальцев, хотя от одного из них мы уже получили ACTION_UP.

В последней паре записей в листинге 16.11 мы видим еще одно событие ACTION_MOVE для второго пальца, за которым следует событие ACTION_UP для второго пальца. На этот раз значение действия равно 1 (ACTION_UP). Значение действия 262 мы не получим, почему — будет объяснено ниже. Отметим также, что в нашем событии ACTION_MOVE индекс указателя изменяется с 1 на 0, но ID указателя остается равен 1.

Вернемся к началу листинга 16.11, где первый палец опущен и имеет ID указателя 0. Почему же у нас не получается значение действия 0x00000005 (или 5 в десятичной системе), если первый палец прикоснулся к экрану раньше всех остальных? Это хороший вопрос, на который нет простого ответа. Значение действия 5 можно было бы получить при следующем сценарии: дотрагиваемся экрана первым пальцем, затем вторым пальцем, получаем значения действия 0 и 261 (пока игнорируем значения ACTION_MOVE). Теперь поднимаем первый палец (значение действия 6) и снова опускаем его на экран. ID указателя второго пальца (второй палец) остается равен 1. В тот момент, когда первый палец был в воздухе, программе было известно только об ID указателя со значением 1. Когда первый палец вновь коснулся экрана, Android присвоил ID указателя 0 первому пальцу, а поскольку уже известно, что с устройством работает несколько пальцев, мы получаем значение действия 5 (ID указателя равно 0 и значение действия равно 5). Ответ на вопрос — «обратная совместимость», но это не самый хороший ответ. В ситуации с двумя пальцами, если первый палец касается экрана в определенном месте, а второй — в другом месте, программа, неспособная к работе с мультитач, не сможет уловить, когда будет поднят с экрана первый палец. Это объясняется тем, что отрыв первого пальца даст значение действия 6, а не 1. Только когда будет поднят и второй палец, такая программа получит значение действия 1.

Когда на экране остается только один палец, Android расценивает ситуацию как одиночное касание. Поэтому получается «старое» значение ACTION_UP, равное 1, как при одиночном прикосновении, а не 6, как при технологии мультитач, вместе с ID указателя. Но подождите, ID указателя последнего пальца на экране в нашем предыдущем примере был равен 1, поэтому мы на самом деле должны были получить значение 262. В коде такие случаи требуют особого внимания. Если ID указателя

равен 0, результирующее значение ACTION_DOWN может быть равно 0 или 5, в зависимости от того, какие указатели при этом используются. Последний палец получит значение ACTION_UP, равное 1, независимо от того, каким будет ID указателя.

Чтобы было проще понять, что происходит, в классе MotionEvent предусмотрено несколько вспомогательных констант. Например, MotionEvent.ACTION_POINTER_3_DOWN равно 0x00000205 (или 517 в десятичной системе) — этот случай был описан выше, когда на экран был опущен третий палец. Возможно, данные значения и не очень-то полезны, лучше сверяться с указателем ID во втором байте и с действием в первом байте. Но на самом деле даже лучше будет использовать некоторые другие константы из класса MotionEvent для считывания значения, возвращаемого getAction(). Имеются в виду константы MotionEvent.ACTION_POINTER_ID_MASK, MotionEvent.ACTION_MASK и MotionEvent.ACTION_POINTER_ID_SHIFT. Сочетая возвращенное значение с каждой из этих масок и соотнося результаты с ID указателей, вы сможете с достаточной точностью восстановить ситуацию, независимо от того, с каким количеством пальцев может взаимодействовать устройство. Пример соответствующего кода приведен в листинге 16.12.

Листинг 16.12. Образец кода для уточнения результата, получаемого от MotionEvent.getAction()

```
int action = event.getAction();
int ptrId = event.getPointerId(0);
if(event.getPointerCount() > 1)
   ptrId = (action & MotionEvent.ACTION_POINTER_ID_MASK) >>>
                          MotionEvent.ACTION_POINTER_ID_SHIFT;
action = action & MotionEvent.ACTION_MASK;
if(action < 7 && action > 4)
   action = action - 5;
int ptrIndex = event.findPointerIndex(ptrId);
```

Обратите внимание: данный код обрабатывает вышеописанные странности, когда ID указателя последнего пальца, оставшегося на экране, не входит в значение, возвращаемое в методе getAction(), а также случай, когда часть значения, соответствующая действию, равна 5 или 6, а не 0 или 1. После того как эти инструкции (statements) в листинге 16.12 будут выполнены, ptrId будет содержать ID указателя, связанный с действием, action — значение от 0 до 4 включительно, а ptrIndex — индексное значение указателя для работы с getX() и другими подобными методами MotionEvent. Один из способов рассмотрения значений, возвращаемых getAction(), — придерживаться правила, согласно которому любое значение более 4 относится к ID указателя. Любое значение, меньшее или равное 4, относится к известному нам конкретному пальцу, независимо от того, каков ID указателя.

Использование касаний при работе с картами

Касания могут применяться и при работе с картами. Вы уже видели, как прикосновение к карте может вызывать изменение масштаба либо позволяет панорамировать карту в режиме «вид сбоку». Это встроенные функции карт. Но что делать, если нам требуется нечто иное? Мы покажем, как реализовать при работе с картами

один интересный набор функций, в том числе дать пользователю возможность выбирать точку щелчком и узнавать ее широту и долготу. Из этого можно извлечь немало пользы.

Один из основных классов для работы с картами — MapView. В нем есть метод onTouchEvent(), точно как у видов, которые мы рассмотрели выше. Единственный аргумент, который может приниматься этим методом, — MotionEvent. Кроме того, можно использовать метод setOnTouchListener(), задающий обработчик обратных вызовов для событий касания, совершаемых в MapView. Другие основные типы объектов на картах — это набор Overlay, в том числе ItemizedOverlay и MyLocationOverlay. Обо всем этом было рассказано в главе 7. Классы Overlay также используют метод onTouchEvent(), но его характеристики немного непохожи на качества onTouchEvent(), используемого с обычным View. В случае с Overlay метод такой:

```
onTouchEvent(android.view.MotionEvent e, MapView mapView)
```

Если мы хотим осуществлять с картами другие операции, то можем переопределить метод onTouchEvent(). Методы чаще переопределяются в классе Overlay, чем в MapView, и в этом разделе акценты будут расставлены соответствующим образом. Как и раньше, метод onTouchEvent() в классах Overlay работает с объектами MotionEvent. При работе с картами объект MotionEvent также содержит координаты X и Y той точки, в которой пользователь коснулся экрана. При работе с картами польза от этого очень снижается, так как обычно на карте требуется совершенно точно знать то место, которого коснулся пользователь. Но и для этого есть способы.

В MapView есть интерфейс, называемый Projection, а в состав Projection входят методы, позволяющие конвертировать пикселы в GeoPoint или GeoPoint в пикселы. Для получения Projection вызывается метод MapView.getProjection(). Имея Projection, для преобразования можно использовать методы fromPixels() и toPixels(). Помните: Projection хорош только в тех случаях, когда карта «стоит на месте» и не изменяется. В методе onTouchEvent() можно преобразовывать значения местоположения X и Y в GeoPoint при помощи fromPixels().

Интересный и очень полезный метод класса Overlay — onTap(), подобный методу onTouch(), рассмотренному выше в этой главе, но отличающийся одной ключевой особенностью. В Overlay географических карт нет метода onTouch(). А метод onTap() имеет следующую характеристику:

```
public boolean onTap(GeoPoint p, MapView mapView)
```

Это означает, что когда пользователь касается Overlay, метод onTap() вызывается с GeoPoint той точки, которой коснулся пользователь. Так экономится масса времени, поскольку уже точно известно, где произошло касание. Больше не нужно беспокоиться о преобразовании координат X и Y в координаты широты и долготы. Android делает это за нас.

Теперь давайте заново рассмотрим пример из главы 7, в котором мы делали карту с кнопками для работы в различных режимах (**Satellite** (Спутник), **Street** (Улицы), **Traffic** (Трафик) и **Normal** (Обычный)). Мы собираемся добавить возможность просмотра StreetView определенного места с карты. Чтобы это сделать, мы должны добавить к MapView объект Overlay, а когда на этом Overlay произойдет событие касания, мы запускаем намерение, инициирующее StreetView этой точки. Сначала сделаем в Eclipse копию MapsDemo из главы 7 (см. листинги 7.12 и 7.13).

Затем воспользуемся листингом 16.13 и изменим метод onCreate() основного Activity, а также добавим новый класс в файле ClickReceiver.java, также имеющемся в этом листинге. Изменения, внесенные в метод onCreate(), показаны полужирным. Пользовательский интерфейс будет выглядеть так же, как на рис. 7.7.

Листинг 16.13. Добавление работы с касаниями в пример MapsDemo

```
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.mapview);

      mapView = (MapView)findViewById(R.id.mapview);

      ClickReceiver clickRecvr = new ClickReceiver(this);
      mapView.getOverlays().add(clickRecvr);
   }

// это файл ClickReceiver.java
import android.content.Context;
import android.content.Intent;
import android.net.Uri;
import android.util.Log;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

public class ClickReceiver extends Overlay{
   private static final String TAG = "ClickReceiver";
   private Context context;

   public ClickReceiver(Context _context) {
      context = _context;
   }

   @Override
   public boolean onTap(GeoPoint p, MapView mapView) {
      Log.v(TAG, "Received a click at this point: " + p);

      if(mapView.isStreetView()) {
         Intent myIntent = new Intent(Intent.ACTION_VIEW, Uri.parse
            ("google.streetview:cbll=" +
            (float)p.getLatitudeE6() / 1000000f +
            "," + (float)p.getLongitudeE6() / 1000000f
            +"&cbp=1,180,,0,1.0"
            ));
         context.startActivity(myIntent);
            return true;
      }
      return false;
   }
}
```

Вот и все, что нам нужно, чтобы наш новый пример заработал — если, конечно, StreetView не содержится у вас в эмуляторе или в устройстве по умолчанию. StreetView был включен в эмуляторы для CupCake (1.5) и Donut (1.6), но удален из эмулятора Eclair (2.0). Один из способов обойти эту проблему — работать с реальным устройством, на котором должен быть установлен StreetView, и тестировать все там. Если вы все же работаете с эмулятором, можете попробовать следующую простую процедуру.

1. Настроить виртуальное устройство Android (AVD), которое базируется на API Google, версии 1.6 или 1.5.
2. Применить пакет `adb pull /system/app/StreetView.apk`, чтобы скопировать это приложение из эмулятора на жесткий диск реального устройства.
3. Настроить AVD, базирующееся на Google API для той версии, с которой вы хотите работать.
4. Применить `adb install StreetView.apk` с файлом APK, скопированным в шаге 2.

Таким образом, программа StreetView должна установиться в эмулятор и приведенный выше пример будет работать.

Запустив модифицированное приложение MapsDemo, увеличьте масштаб города так, чтобы были видны улицы. Нажмите кнопку **Street** (Улицы), чтобы на тех улицах, где поддерживается StreetView, появились голубые контуры (имеются в виду улицы, фотографии которых есть в базе данных Google). Теперь можно коснуться изображения улицы — будет вызван метод `onTap()` класса `ClickReceiver`, который, в свою очередь, свяжется с явлением StreetView с точкой, в которой произошло событие касания, — это делается при помощи намерения. Если коснуться на карте такой точки, для которой в StreetView нет изображений, отобразится пустой экран StreetView с замечанием типа **Invalid panorama** (Неправильная панорамара). Это означает, что Google не удается найти изображений, снятых поблизости от данной точки. Нажмите стрелку **Back** (Назад), чтобы вернуться в программу Maps, и попробуйте просмотреть другое место. Если вы просмотрите LogCat, то заметите, что там записаны данные о широте и долготе тех точек, в которых произошло касание. Обратите внимание: объект `GeoPoint` использует с широтой и долготой единицы `ints` (натуральные числа), тогда как URI StreetView требует значения в формате `floats` (числа с плавающей точкой).

В нашем демонстрационном приложении мы решили посылать намерение с широтой и долготой точки прикосновения к явлению StreetView. Но можно представить себе и другие варианты. Имея широту и долготу места на карте, можно при помощи `GeoCoder` узнать, что находится вокруг. Местоположение можно использовать, чтобы перейти к нему при помощи пошаговых инструкций (turn-by-turn). Можно измерить, насколько это место удалено от точки, в которой мы находимся. Мы даже можем сохранить эту точку и поработать с ней позже.

Жесты

Жесты — это особые события, происходящие на сенсорном экране. В принципе, жест можно определить как заранее заданное движение пальцев по экрану, которое

программа ожидает от пользователя. Если движение пользователя и заданный жест совпадают, программа может запускать специальную логику, в зависимости от того, что данный жест означает в вашей программе. Для работы с жестами нужен наложенный слой, который может идентифицировать серии движений, совершаемые пользователем, и передавать их базовому явлению. Если в пользовательском интерфейсе применяются жесты, сам он значительно упрощается, так как функции кнопок и других элементов управления можно выполнять пальцами или как будто рисуя. Жесты позволяют создавать очень интересные игровые интерфейсы. В этом разделе мы научимся записывать жесты и использовать их в нашей программе.

Прежде чем заняться кодом для жестов, поиграем с программой Gestures Builder, которая по умолчанию есть в эмуляторе. Так вы лучше поймете, что же такое жест. Gestures Builder создает специальный файл, в котором содержится библиотека жестов, и управляет им. Запустите эмулятор в Eclipse, разблокируйте устройство с эмулятором, перейдите в раздел apps (Приложения) и откройте Gestures Builder. На рис. 16.4 показан его значок.



Рис. 16.4. Значок Gestures Builder

Обычно при запуске Gestures Builder открывается пустой экран. Нажмите кнопку Add gesture (Добавить жест). Вам будет предложено указать Name (Имя). Жест, который вы собираетесь записать, будет сохранен под этим именем. Позже это имя будет использоваться при ссылке на жест и станет своего рода именем команды. Когда пользователь совершает в вашей программе жест, имя этого жеста передается методам программы, которые могут интерпретировать действия пользователя. Название жеста может быть существительным и свидетельствовать о его форме — например, Checkmark (Галочка) или Spiral (Спираль). Либо жест можно обозначить как команду, скажем, Fetch (Выбрать) или Stop (Стоп). Назовем наш первый жест «галочка» и укажем его имя — checkmark. Теперь на чистом экране, который открылся чуть раньше (см. выше), нарисуем галочку. Если первый вариант вам не понравится — просто перерисуйте. Новый вариант автоматически заменяет более ранний. Когда галочка вас устроит, нажмите Done (Готово). Галочка должна выглядеть примерно как на рис. 16.5.

Обратите внимание — можно записать различные виды галочек и все их назвать checkmark. Запишите еще какой-нибудь жест, похожий на галочку, и также назовите его checkmark. Эта фигура может быть больше или меньше, чем ваша первая галочка, либо еще как-то от нее отличаться, но в целом, ее контуры будут именно как у галочки. Поэкспериментируйте с кнопкой Add Gesture (Добавить жест) и добавьте еще несколько жестов с другими названиями. Каждый раз при нажатии кнопки Done (Готово) в библиотеку будет добавляться новый жест. Попробуйте записать мультитач-жесты, например провести по экрану двумя пальцами одновременно

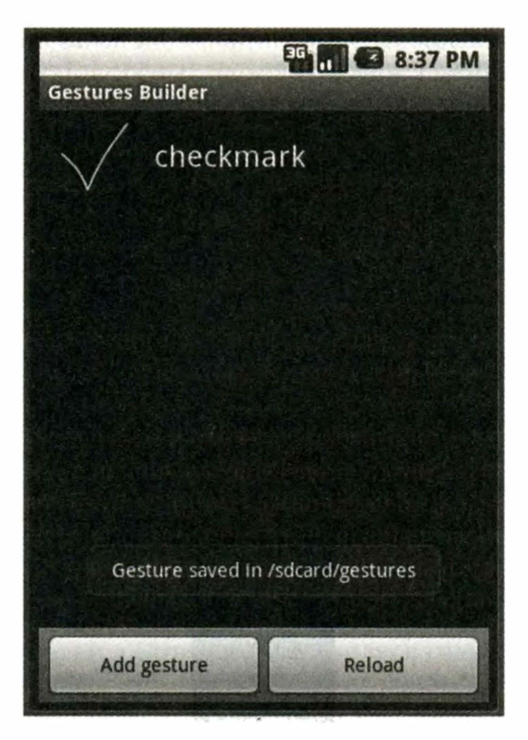


Рис. 16.5. Жест «галочка», сохраненный в /sdcard

и сделать знак равенства. В Android 2.0 такая функция отсутствует, поэтому у вас получится только одна линия. Возможно, в будущих версиях жесты, проводимые одновременно двумя и более пальцами будут поддерживаться.

Каждый жест имеет название и состоит из штрихов. Штрих жеста (gesture stroke) – это последовательность касаний, начинающаяся с прикосновения к экрану и завершаемая с отрывом пальца. Выше говорилось о том, что последовательность касаний состоит из серии объектов `MotionEvent`. Подобным образом жест состоит из жестовых точек. Жесты собираются в специальном хранилище (gesture store). Одна библиотека жестов содержит одно хранилище. В Android все эти сущности представлены в виде классов, которые можно использовать в коде. На рис. 16.6 приведена схема, иллюстрирующая эти отношения.

При создании жестов мы и не можем использовать мультитач, но есть способ объединять в жесте несколько штрихов. Например, чтобы создать жест в форме буквы «Е», придется провести как минимум два жеста: одним нарисовать верхнюю, вертикальную и нижнюю линии, а вторым – центральную линию. Другой способ – нарисовать вертикальную линию, а потом три отдельные горизонтальные линии. «Е» можно нарисовать и другими способами и, что не может не радовать, библиотека жестов позволяет сохранять все эти варианты под одним и тем же названием. Попробуйте записать «Е» несколькими способами, поставив себя на место пользователя. Вы убедитесь: программа понимает, что каждый из вариантов – это именно «Е». На рис. 16.7 показаны различные варианты записи «Е».

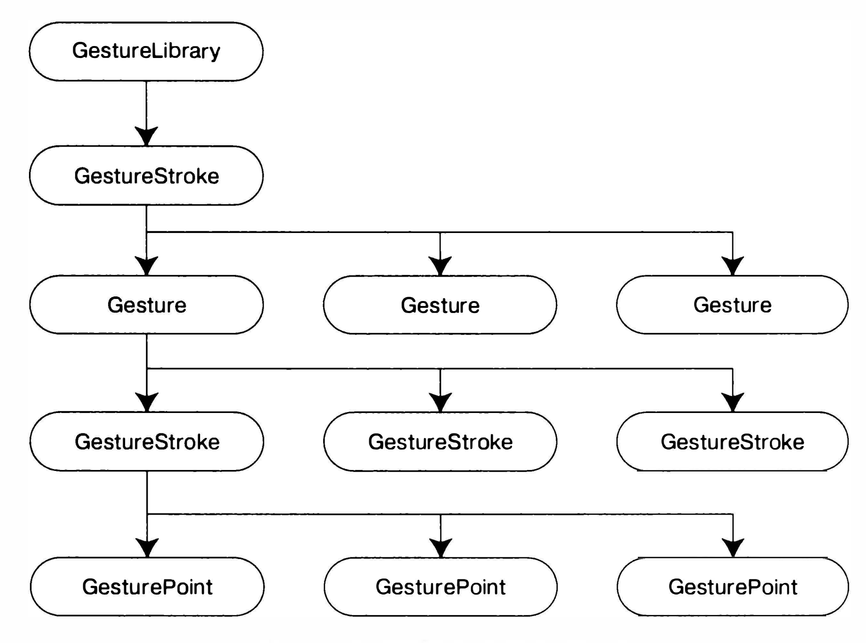


Рис. 16.6. Структура жестовых классов

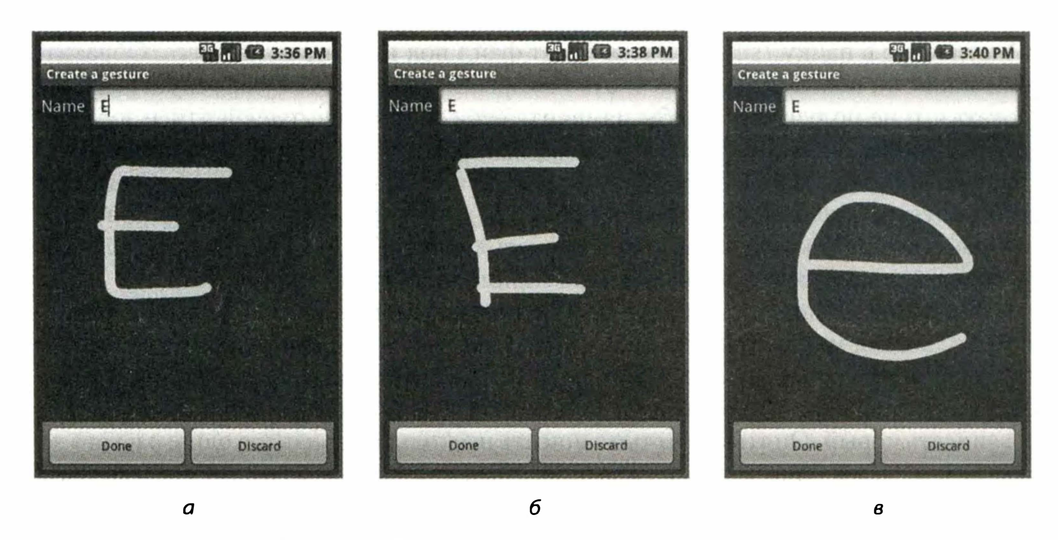


Рис. 16.7. Различные варианты записи жеста «E»

Возможно, у вас возникнут сложности при рисовании многоштриховых жестов в эмуляторе (то есть в Gestures Builder). Как было указано выше, можно просто нарисовать новый вариант жеста поверх старого, и старый вариант жеста будет стерт. А как Android отличает случаи, в которых начинается рисование нового жеста от ситуаций, когда просто добавляется новый штрих к имеющемуся жесту? В Android применяется особое значение `FadeOffset`. Это значение в миллисекундах,

и если время ожидания перед рисованием нового штриха превышает заданную здесь величину, система записывает новый жест. По умолчанию FadeOffset имеет значение, равное 420 миллисекундам. Это означает, что если вы рисуете на экране жест и в ходе этого отрываете палец от экрана больше чем на 420 миллисекунд, Android сочтет, что жест готов, и сохранит в качестве нового жеста уже имеющиеся на экране штрихи. В реальном устройстве по умолчанию может быть задано время, достаточное для того, чтобы начать рисовать следующий штрих. Но в эмуляторе дело обстоит иначе — здесь на процесс непосредственно влияет скорость рабочей станции.

Если при работе в эмуляторе у вас возникают проблемы с Gestures Builder (не получается рисовать многоштриховые жесты), создайте собственную версию Gestures Builder и измените стандартное значение FadeOffset. Образец Gestures Builder содержится в каталоге с Android SDK по адресу platforms/android-2.0/samples/GestureBuilder. В Eclipse используйте команду **Create project from existing sample** (Создать проект на базе имеющегося образца). Здесь из раскрывающегося списка выберите **Gestures Builder**. Затем перейдите к файлу /res/layout/create_gesture.xml данного проекта и добавьте к элементу GestureOverlayView атрибут android:fadeOffset="1000". Таким образом, FadeOffset увеличится до 1 секунды (1000 миллисекунд). Можете задать и любое другое значение.

Теперь разберемся, где сохраняются готовые жесты. Специальное всплывающее сообщение Gestures Builder подсказывает, что жесты попадают в /sdcard/gestures. При помощи **File Explorer** (Проводник по файлам) в Eclipse либо adb перейдите в эмуляторе в папку /sdcard. Здесь будет файл под названием gestures. Обратите внимание: он невелик. Gestures — это бинарный файл, поэтому вручную редактировать его не получится. Чтобы изменить его содержимое, следует пользоваться программой Gestures Builder. Создавая программу, в которой поддерживаются жесты, нужно скопировать файл gestures в каталог программы /res/raw. Для этого используется функция копирования (**File Copy**) программы **File Explorer** (Проводник по файлам) либо команда pull из adb — эта команда переносит файл gestures на рабочую станцию, а отсюда уже можно скопировать его в проект.

В Gestures Builder можно не только добавлять новые жесты, но и работать со специальными меню. Для перехода в такое меню нужно сделать длинный щелчок на названии жеста. В этом меню можно изменить название жеста либо удалить его. Перезаписать жест нельзя, то есть, если жест вас не устраивает, его нужно удалить и записать новый. Опять же можно записать несколько вариантов одного и того же жеста и дать им одно и то же название. Название не обязательно должно быть уникальным, однако жесты с одинаковым названием следует делать похожими. Так можно учитывать различные способы, которыми пользователь может выполнять один и тот же жест. Например, можно записать несколько разных галочек, и дать им всем имя checkmark. Когда пользователь рисует в вашей программе галочку, то, если рисунок совпадает с одним из записанных вами вариантов, программа получит сообщение о том, что в ней рисуют галочку.

Теперь напишем программу, в которой используем наш новый файл gestures. В Eclipse создадим новый проект Android. В листинге 16.14 приведен XML-шаблон нашего файла и код класса Activity.

Листинг 16.14. Код Java для программы, работающей с жестами

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Это файл /res/layout/main.xml -->
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   >
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="Draw gestures and I'll guess what they are"
   />

<android.gesture.GestureOverlayView
   android:id="@+id/gestureOverlay"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:gestureStrokeType="multiple"
   android:fadeOffset="1000" />

</LinearLayout>

import java.util.ArrayList;
import android.app.Activity;
import android.gesture.Gesture;
import android.gesture.GestureLibraries;
import android.gesture.GestureLibrary;
import android.gesture.GestureOverlayView;
import android.gesture.Prediction;
import android.gesture.GestureOverlayView.OnGesturePerformedListener;
import android.os.Bundle;
import android.util.Log;
import android.widget.Toast;

public class MainActivity extends Activity implements
                OnGesturePerformedListener {
   private static final String TAG = "Gesture Revealer";
   GestureLibrary gestureLib = null;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
// gestureLib = GestureLibraries.fromRawResource(this, R.raw.gestures);
      gestureLib = GestureLibraries.fromFile("/sdcard/gestures");
      if (!gestureLib.load()) {
         Toast.makeText(this, "Could not load /sdcard/gestures",
            Toast.LENGTH_SHORT).show();
```

```
            finish();
        }

        // Рассмотрим библиотеку жестов, с которой мы работаем.
        Log.v(TAG, "Library features:");
        Log.v(TAG, " Orientation style: " +
                gestureLib.getOrientationStyle());
        Log.v(TAG, " Sequence type: " + gestureLib.getSequenceType());
        for( String gestureName : gestureLib.getGestureEntries() ) {
            Log.v(TAG, "For gesture " + gestureName);
            int i = 1;
            for( Gesture gesture : gestureLib.getGestures(gestureName) ) {
                Log.v(TAG, " " + i + ": ID: " + gesture.getID());
                Log.v(TAG, " " + i + ": Strokes count: " +
                        gesture.getStrokesCount());
                Log.v(TAG, " " + i + ": Stroke length: " +
                        gesture.getLength());
                i++;
            }
        }

        GestureOverlayView gestureView =
            (GestureOverlayView) findViewById(R.id.gestureOverlay);
        gestureView.addOnGesturePerformedListener(this);
    }

    @Override
    public void onGesturePerformed(GestureOverlayView view,
                Gesture gesture) {
        ArrayList<Prediction> predictions = gestureLib.recognize(gesture);

        if (predictions.size() > 0) {
            Prediction prediction = (Prediction) predictions.get(0);
            if (prediction.score > 1.0) {
                Toast.makeText(this, prediction.name,
                            Toast.LENGTH_SHORT).show();
                for(int i=0;i<predictions.size();i++)
                    Log.v(TAG, "prediction " + predictions.get(i).name +
                        " - score = " + predictions.get(i).score);
            }
        }
    }
}
```

В данном примере мы просто получаем доступ к тому самому файлу, который был создан программой Gestures Builder. Для этого вместе с методом onCreate() используем метод GestureLibraries.fromFile(). Однако файл также содержит и комментарии о том, как попасть в файл с жестами, являющийся частью вашей программы. Если бы вы использовали метод fromRawResource(), то в качестве аргумента он получал бы регулярный ID ресурса, а файл с жестами находился бы в каталоге /res/raw.

Получившаяся у нас программа не так уж и многофункциональна, но, запустив ее, можно лучше понять, что происходит внутри системы Android в процессе рисования жестов. Сначала программа загружает файл с жестами и записывает в файл регистрации, что именно она нашла. Кроме того, в файле регистрации сохраняется информация о том, как программа пытается сопоставить жесты, рисуемые на экране, с имеющейся информацией. Далее запустим программу Gesture Revealer — при этом предполагается, что Gestures Builder уже работает, а в файле /sdcard/gestures уже записано несколько жестов. Посмотрите, как для каждого жеста регистрируются ID, количество штрихов и длина.

Нарисуйте на экране жест, который есть в библиотеке жестов. Затем нарисуйте жест, которого там точно нет. Посмотрите записи LogCat. Вы заметите, что иногда рисуете жест, который должен присутствовать в библиотеке, но система не распознает его, либо вы рисуете один жест, а Android принимает его за другой. Правда, обычно она интерпретирует жесты правильно. Кроме того, обратите внимание: когда Android распознает ваш жест, список имеющихся в библиотеке жестов обновляется, но если Android не понимает, что за жест вы ввели, в predictions ничего не отобразится.

Кроме того, посмотрим, как обрабатываются жесты, состоящие из нескольких штрихов, например «Е», а также что происходит, если при этом в рисовании штрихов вы делаете достаточно большие промежутки. Программа несколько раз сохранит частично выполненный рисунок и сравнит такие недоделанные варианты с содержимым библиотеки жестов. В результате получим неполное совпадение (wrong match) либо вообще никакого совпадения. Такие задержки при рисовании контролируются с помощью FadeOffset. Сложности возникают по следующим причинам: Android начинает сопоставлять выполняемые жесты с имеющимися образцами, как только мы нарисуем жест (если только мы не подождем немного и не начнем рисовать новый штрих). Следовательно, FadeOffset выполняет две задачи: во-первых, определяет временной интервал, по истечении которого новый штрих будет считаться уже частью нового жеста, а во-вторых — устанавливает период ожидания, после которого можно начинать сравнивать новый жест с образцами жестов, имеющимися в библиотеке. Если сделать FadeOffset очень длительным, придется долго ждать, пока система начнет сравнивать начертанный нами жест с образцами. Если, напротив, чрезмерно сократить FadeOffset, нам будет сложно рисовать многоштриховые жесты, так как Android будет считать, что мы уже закончили жест, когда мы только переходим к новому штриху. Подходит ли вам стандартное значение 420 миллисекунд — решайте сами. Можно задать значение Preference, чтобы пользователь мог самостоятельно настроить этот период.

Обсуждая многоштриховые жесты, необходимо отметить, что GestureOverlayView имеет настройку, определяющую, будет система ожидать многоштриховые жесты либо нет. Соответствующий атрибут Android — android:gestureStrokeType. Он может принимать значения single либо multiple. Если вы собираетесь работать с многоштриховыми жестами, этот атрибут необходимо установить. Его можно задать программным способом — при помощи метода setGestureStrokeType(int type), применяя в качестве аргумента GestureOverlayView.GESTURE_STROKE_TYPE_SINGLE или GestureOverlayView.GESTURE_STROKE_TYPE_MULTIPLE. GestureOverlayView также имеет XML-атрибуты и методы, позволяющие определять цвет или толщину линий.

Создавая собственную программу, приспособленную для работы с жестами, нужно решить, с какими жестами она будет работать, создать библиотеку этих жестов, а затем задействовать интерфейс onGesturePerformedListener, например, в классе Activity, который позволит распознавать жесты и предпринимать необходимые действия.

Что делать, если мы хотим дать пользователям возможность записывать собственные жесты? Допустим, пользователь решит применить свой жест для определенного действия, а вы уже задали для этого действия стандартный жест. Такое случается, а это значит, что нужно предусмотреть функцию записи дополнительной информации в библиотеку жестов. Эту функцию логично разместить в /sdcard. А проще всего создать новый файл библиотеки жестов, считать стандартные жесты из файла библиотеки, прилагаемого к вашей программе, и при необходимости заменять стандартные жесты теми, для которых у пользователя есть свои варианты. Можно применять описанную выше реализацию Gestures Builder и с ее помощью опробовать внедрение механизма для записи жестов. Либо можно создать новую, открытую для записи библиотеку и заносить в нее только пользовательские жесты, а затем загрузить в программу обе библиотеки — пользовательскую и оригинальную. В методе onGesturePerformed() можно задействовать recognize(), применяя его сначала к пользовательской библиотеке, а затем к стандартной. Можно сравнивать варианты из верхних частей списков (predictions) каждой библиотеки и решать, какое действие предпринять.

Резюме

В этой главе было рассказано, как работать с сенсорными экранами. Сначала мы изучили варианты приложений с обработкой одиночных касаний, затем — технологию мультитач. Мы показали, как касания применяются при работе с картами и какие удобные классы и методы предоставляются в Android именно для картографических приложений. Наконец, мы изучили, как в Android построена работа с жестами, то есть рассмотрели способ, позволяющий обрабатывать пользовательский ввод новым и гораздо более простым образом, нежели при помощи виртуальной клавиатуры или других элементов пользовательского интерфейса.

17 Titanium Mobile: разработка для Android на основе WebKit

В этой главе будет дана вводная информация об инновационном комплексном подходе к программированию приложений для платформы Android. Он относится к тенденции, начавшейся в IT-технологиях с появлением RIA (Rich Internet Applications, «насыщенные», или «богатые», интернет-приложения). К ключевым особенностям RIA относятся широкое применение функции перетаскивания (Drag and Drop), анимация, взаимодействие с серверами без обновления, при помощи HTML-браузеров. Хотя такие функции обычно выполняются при помощи плагинов, например Flash, последние достижения в области RIA позволяют реализовывать их, пользуясь преимуществами объектной модели документа (Document Object Model, DOM) в HTML.

Продукт Titanium Mobile, разработанный силами компании Appcelerator Inc. (http://www.appcelerator.com), которая адаптирует достижения RIA для мобильных устройств, не просто позволяет применять функции RIA на популярных мобильных платформах (Android и IPhone), но к тому же является свободным ПО и распространяется по лицензии Apache v2.0. В этой главе мы познакомим вас с этой новой, но уже известной парадигмой, опишем архитектуру и механизм работы Titanium Mobile.

В этой главе три основных раздела. Сначала будет дан обзор Titanium Mobile и описана его история, архитектура и среда разработки. Мы покажем, как подписаться на Titanium Mobile и скачать его, расскажем о компонентах Titanium Developer, в том числе об изолированной программной среде, так называемой «песочнице» (sandbox), где вы сможете ввести и протестировать несколько строк кода на JavaScript, например `Hello world`.

Во втором разделе на примере `Hello world` будет рассмотрен жизненный цикл проекта. С этим примером мы будем работать уже не в «песочнице»; проект будет иметь формальную структуру, которую вы сможете построить и распространить. Во втором разделе будет создан новый проект, потом мы протестируем его в эмуляторе, сохраним в пакете APK и подпишем пакет так, чтобы его можно было устанавливать в других эмуляторах или устройствах.

В третьем разделе мы научимся писать клиентские приложения на JavaScript, не пользуясь серверными фреймворками для пользовательских интерфейсов (в частности, JSP — серверные страницы Java) или ASP.NET. При рассмотрении этой темы мы изучим продвинутый вариант JavaScript, без которого нам не обойтись. Кроме того, мы поговорим об одной из самых важных библиотек JavaScript — jQuery. Будут также перечислены обертки, предназначенные для внедрения API JavaScript на нативную платформу Android. Эти обертки входят в состав Titanium Mobile. В заключение главы мы объясним важность подхода с применением оберток при разработке в Android.

ПРИМЕЧАНИЕ

Занимаясь разработкой программ для Android по методу Titanium, вы сможете научиться более быстрому и красивому подходу к работе. Его красота обусловлена простотой и гибкостью, характерной для пользовательских интерфейсов, в которых применяются HTML и CSS. И это уже не говоря о предоставляемой Titanium Mobile платформонезависимой абстракции, благодаря которой ваши программы смогут работать на самых разных мобильных платформах.

Начиная работу, еще раз оговоримся, что эта глава служит лишь введением в Titanium Mobile, мы не будем рассматривать его в подробностях. Однако в этой главе вы сможете составить точную карту открывающегося перед вами лабиринта инструментов open source, которые понадобятся вам для работы с этой парадигмой. Это очень важно, поскольку документация по Titanium Mobile предполагает, что специалист уже знаком с принципами разработки программ в Web 2.0.

Обзор Titanium Mobile

Если вы занимаетесь веб-разработкой, то, вероятно, уже знакомы с некоторыми технологиями и инструментами, применяемыми в браузерах и области RIA, в частности Flash/Flex от Adobe, Silverlight от Microsoft, JavaFx от Sun и Laszlo от Laszlo Systems.

Эти инструменты обеспечивают насыщенное взаимодействие пользователя с программой, предоставляя такие возможности, как анимация, перетаскивание, элементы управления деревьями (tree controls) и насыщенные таблицы (richer tables).

Во Flash/Flex эти функции реализуются при помощи браузерного плагина Flash. В Silverlight для той же цели применяется браузерный плагин Dotnet, предоставляющий CLR (общеязыковая среда исполнения). В JavaFx аналогичные задачи выполняются посредством JRE (среда исполнения Java). Laszlo также использует плагин Flash.

В сфере RIA есть одна технология, которая обходится без всяких плагинов, а применяет нативные браузерные элементы управления. В этой альтернативной технологии используются библиотеки JavaScript, в которых применяются преимущества объектной модели документа, действующей в HTML, и программирование насыщенных интернет-приложений происходит непосредственно, без подключения дополнительных модулей. Библиотеки JavaScript обеспечивают создание архитектуры, которая исключительно хорошо подходит для создания насыщенных приложений с применением объектной модели документа (DOM) и асинхронного JavaScript и XML (Ajax).

ПРИМЕЧАНИЕ

Если вы не знаете, что такое DOM и Ajax, почитайте об этом в специальных источниках, чтобы разобраться, какую роль эти технологии играют при создании насыщенных интернет-приложений.

Подход к созданию RIA, основанный на применении JavaScript и HTML, полностью без плагинов, поднимает нас на такой уровень разработки, на котором мы можем идти в ногу со временем и одновременно обойтись без предварительного скоростного заучивания сложного материала.

Да, может сказать читатель, все это очень актуально для веб-разработки, но как эти вещи связаны с Android, если мы говорим о нативных приложениях? Оказывается, браузер Android основан на Chrome, который, в свою очередь, использует вездесущий движок WebKit — основу браузеров Chrome и Safari.

Появляются технологии, обеспечивающие нативное управление WebKit при помощи HTML- и JavaScript-файлов, которые сохранены на локальном устройстве. Именно здесь вступает в игру Titanium Mobile. В Titanium Mobile задействуется WebKit, обеспечивающий создание кросс-платформенных решений, которые работают как на ПК с Windows и Mac, так и на мобильных устройствах IPhone и Android.

ЗАМЕЧАНИЕ

Компания Appcelerator Inc. (первоначальное название — Hakano) расположена в Маунтин-Вью, штат Калифорния. Она была основана в 2006 году Джеффом Хейни (Jeff Haynie) и Ноланом Райтом (Nolan Wright) и предлагает продукты из области Web 2.0. В начале 2008 года они расширили свою стратегию развития и стали использовать кросс-платформенный, основанный на WebKit подход, обеспечивающий разработку одновременно для ПК и мобильных платформ. Благодаря их усилиям появилось семейство продуктов Titanium.

Теперь изучим, каков Titanium Mobile изнутри и как он обеспечивает богатое взаимодействие с пользователем, а также кросс-платформенную совместимость.

Архитектура

По существу, Titanium Mobile является оберткой (wrapper) для работы с движком WebKit, используемым в Android и других мобильных устройствах, например IPhone. Для реализации функций WebKit в Titanium Mobile применяется набор API из JavaScript, предназначенных для работы с конкретными библиотеками Android, например медийными или для файловых систем. Такая абстракция, основанная на API из JavaScript и работающая на нативном устройстве, для бывалых веб-разработчиков оказывается универсальным интерфейсом, позволяющим писать приложения для работы в нативной ОС. На схеме (рис. 17.1) в общем показано, как в Titanium выполняются эти задачи.

Данная схема состоит из трех основных разделов (блоков): B1, B2 и B3.

- B1 — это проект, в котором вы разрабатываете определенную программу.
- B2 — это интегрированная среда разработки (IDE) Titanium. Для краткости будем называть ее просто Titanium IDE.
- B3 — это эмулятор или устройство Android.

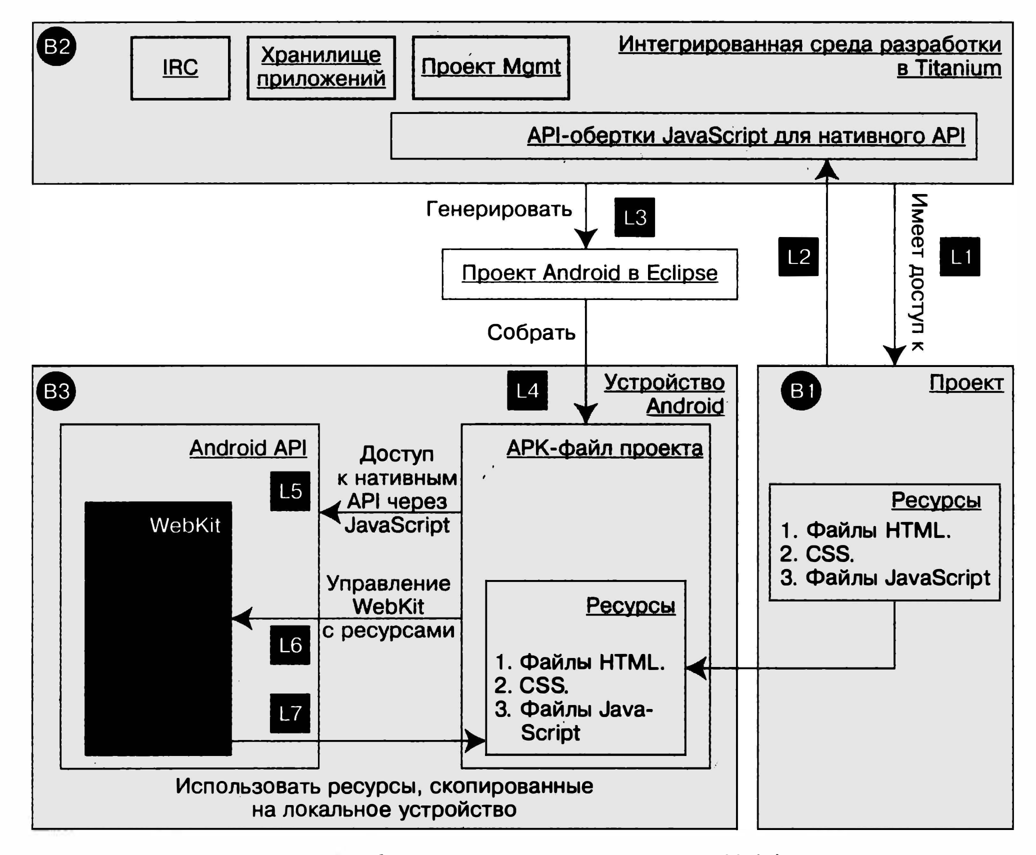


Рис. 17.1. Общая схема архитектуры Titanium Mobile

Проект Titanium, представленный в блоке B1, — это просто директория на жестком диске, в которой хранятся HTML-ресурсы. В отличие от служебных файлов и файлов построения (build files), большинство исходных файлов (source files) располагаются в подкаталоге Resources. Этот каталог проекта известен Titanium IDE (на рис. 17.1 это отношение обозначено линией L1).

JavaScript в этом проекте имеет доступ к нативным библиотекам JavaScript, предоставляемым в Titanium IDE. Этот нативный JavaScript входит в состав Titanium IDE и показан в рамке с Titanium IDE (B2). На рис. 17.1 это отношение обозначено линией L2.

Titanium IDE берет проект и создает файл APK, который затем устанавливается в устройстве (B3). Соответствующие линии — Сгенерировать (L3) и Построить (L4).

ПРИМЕЧАНИЕ

Если быть точными, в IDE создается промежуточный проект (линия L3), который до построения напоминает проект из Eclipse ADT. Это важно знать, так как вы можете взять такой промежуточный проект и отладить его в Eclipse IDE.

После установки указанный файл APK будет управлять WebKit (линия L6) посредством доступных файлов ресурсов. Когда архив APK установлен в устройстве,

файлы из подкаталога Resources (HTML, изображения, JavaScript, CSS и т. д.) также будут скопированы в устройство. Линия L7 показывает, как WebKit находит HTML-файлы с относительными адресами.

Код JavaScript из HTML-файлов (которые сами входят в состав APK) также будет иметь доступ к нативной платформе Android через нативные API JavaScript, предоставляемые в Titanium Developer (линия L5).

Итак, после краткого изучения архитектуры взаимодействий Titanium Mobile и Android, рассмотрим более подробно каждый компонент этой архитектуры:

- проект Titanium, содержащий ресурсы;
- интегрированную среду разработки Titanium (IDE);
- построение и развертывание проекта в устройстве.

Суть проекта в Titanium

Проект Titanium Mobile очень похож на проект для разработки в HTML, где есть файл index.html и подкаталоги, в которых содержатся HTML-файлы, файлы CSS и другие библиотеки JavaScript (как ваши, так и сторонние). В Titanium этот корневой каталог называется Resources, и на вышеприведенной схеме ему соответствует рамка Resources в блоке B1.

При работе с Titanium IDE и начале работы над проектом подкаталог Resources создается автоматически. В примерах, которые приведены далее в этой главе, будет показана точная структура каталогов, используемых в таком проекте. Разработчик может пользоваться любым набором библиотек JavaScript — теми библиотеками, с которыми лучше знаком. Кроме того, есть API, доступ к которым Titanium открывает для кода на JavaScript.

Компоненты интегрированной среды разработки Titanium

Рассмотрим компоненты среды Titanium IDE (на рис. 17.1 это рамка B1). Строго говоря, эта среда совсем не похожа на традиционные среды разработки, например на Eclipse. В ней нет никаких редакторов. Однако в ней можно создать каталоги с проектами и компилировать, строить, тестировать и развертывать проекты. Это в большей степени строительная среда и среда для коллективной работы (collaborative environment). Например, когда вас устроит очередная редакция проекта, можно перейти в Titanium IDE и протестировать программу в эмуляторе Android. Команда Titanium Mobile рекомендует каждому разработчику использовать отдельную IDE для редактирования и работы с JavaScript и HTML.

В состав среды разработки Titanium входят следующие ключевые компоненты:

- *управление проектами* — здесь создаются, строятся, упаковываются, тестируются и развертываются проекты (приложения);
- *хранилище приложений* — хранилище, позволяющее сообществу разработчиков загружать и скачивать свои программы;
- *IRC* — ретранслируемый интернет-чат, где можно обратиться за помощью; он интегрирован непосредственно в ваше рабочее пространство (это действительно удобно).

Далее будут рассмотрены принципы работы каждого компонента.

Построение и развертывание проектов при помощи Titanium IDE

Теперь изучим возможности построения файлов в Titanium IDE. Создавая и редактируя файлы, как и в любом другом проекте, мы приказываем инструменту Titanium Developer построить и протестировать приложение (которое находится в каталоге Resources). Затем IDE преобразует эти файлы в проект Android, структурно это очень похоже на Eclipse ADT.

Затем промежуточный проект Android компилируется, и из него создается файл APK. IDE берет этот APK-файл (точно так же, как Eclipse ADT) и устанавливает его на эмулятор Android. Кроме того, Titanium Developer автоматически активирует эмулятор. Все эти шаги — части этапа «построение и тестирование». По его окончании вы увидите, как ваша программа отобразится в эмуляторе.

Процесс редактирования и тестирования можно повторять до тех пор, пока результат вас не устроит. Затем нужно сделать конечную копию подписанного файла APK, готовую для распространения. (На этапе тестирования подписывать APK-файл не требуется). Titanium Developer IDE делает это автоматически. Однако, чтобы развернуть файл на внешнем устройстве, нужно специально подписать его при помощи Titanium IDE.

При создании APK-файла Titanium Developer копирует каталог Resources в подкаталог с ресурсами (assets subdirectory), так чтобы нативный WebKit имел доступ к этим файлам во время исполнения. Java API также закончат обращаться к нативным версиям API Java из Android.

Как видите, все волшебство Titanium является вполне рациональным. Теперь рассмотрим, что можно сделать при помощи архитектуры WebKit.

Среда разработки Titanium

Кажущаяся тонкость и миниатюрность Titanium Mobile иногда неправильно понимается консервативными разработчиками, привыкшими иметь дело с комплексными решениями, такими как ASP.NET от Microsoft или Flash/Flex от Adobe. В отличие от них, Titanium Mobile основан на открытой системе, из которой и черпает свои силы.

Хотя Titanium Mobile — это всего лишь обертка, в основе его лежат проверенные временем технологии, в частности AJAX, jQuery, DOJO, Mootools, JSON, Aptana и Microsoft Web Express. Titanium Mobile позволяет использовать этот арсенал в Android. Разумеется, Titanium Mobile также привносит дополнительные функции в различные API JavaScript — благодаря набору оберток для работы с базовым API Android.

Работая с Titanium, выберите те из перечисленных инструментов, которые лучше всего подходят для ваших целей. Один инструмент можно взять для пользовательского интерфейса, другой — для обеспечения связи с сервером, а третий — для сохранения параметров.

Выше мы уже говорили о том, что для разработки пользовательских интерфейсов удобно работать с библиотекой jQuery — она проста в изучении, хорошо документирована, позволяет решать при программировании достаточно сложные

задачи. Кроме того, в сообществе разработчиков свободного ПО ожидается, что эта библиотека будет расширяться.

ПРИМЕЧАНИЕ

Ради экономии места мы не будем отдельно рассматривать аспекты сохранения параметров и доступ к серверу в программах Titanium Mobile. Оставляем их вам для самостоятельного изучения. Надеемся, что, описав jQuery и пользовательские интерфейсы, мы сможем показать вам основные принципы разработки в Titanium. Аспекты, не рассмотренные здесь, работают по стандартным принципам и будут понятны интуитивно.

Теперь совершим небольшую экскурсию в Titanium и для начала скачаем Titanium Developer.

Скачивание и установка Titanium Developer

В этом подразделе будет показано, как установить Titanium Mobile, а также будет сделано введение в набор его функций. Мы рассмотрим меню и экраны, имеющиеся в IDE. Вы сможете лучше понять, что можно делать при помощи IDE и как разрабатываются собственные приложения

ПРИМЕЧАНИЕ

Как и в остальных главах этой книги (кроме 2), при написании проектов мы пользуемся операционной системой Windows XP. Обратите внимание: в самом общем смысле в этой теме затрагивается и Mac OS X.

Прежде чем установить Titanium Mobile, на него нужно подписаться. Процесс подписки и установки протекает лучше всего, если вы подключены к Интернету (если же постоянного соединения у вас нет, можно скачать ZIP-архив и установить его). Система рассчитана на работу в подсоединенном режиме, как при установке, так и при разработке (обратите внимание: версия 0.8.1, актуальная на момент выхода книги, должна обеспечивать и настройку прокси-доступа; но авторы не тестировали эту версию).

Подписаться на Titanium Mobile можно по адресу http://www.appcelerator.com. Если не можете найти соответствующую кнопку, то можно подписаться непосредственно по следующей ссылке: http://www.appcelerator.com/products/request-titanium-mobile/.

Когда подписка будет завершена, вы получите по электронной почте письмо с файлом для скачивания и загрузки Titanium Mobile (этот файл довольно велик, около 40 Мбайт). (Когда в процессе установки авторы пытались скачать весь пакет, приходилось обращаться к серверу несколько раз.) После завершения установки создаем ярлык на рабочем столе. Важно отметить, что пока утилиты для удаления для этой программы не существует. Все каталоги при необходимости приходится удалять вручную. В Windows XP для полного удаления Titanium Developer можно сделать следующее (листинг 17.1)

Листинг 17.1. Каталоги, удаляемые при деинсталляции Titanium

```
\documents and settings\all users\application data\Titanium
\documents and settings\\application data\Titanium
\Program Files\Titanium (your install directory)
```

По следующей ссылке даются инструкции по деинсталляции и повторной инсталляции Titanium Developer: http://support.appcelerator.net/faqs/titanium-installation/reinstalling-titanium-developer. По этой же ссылке содержится схожий набор инструкций для Mac OS X.

Ярлык для установленной программы выглядит так, как показано на рис. 17.2.



Рис. 17.2. Ярлык для Titanium Developer

Если нажать этот ярлык, запустится среда разработки Titanium Developer IDE. Пока в ней нет ни одного проекта, Titanium Developer IDE выглядит как на рис. 17.3. Есть два способа создать здесь проект. Можно нажать либо кнопку Create (Создать), либо значок New Project (Новый проект), расположенный сверху. Создание проектов будет подробно рассмотрено ниже, в одноименном подразделе.

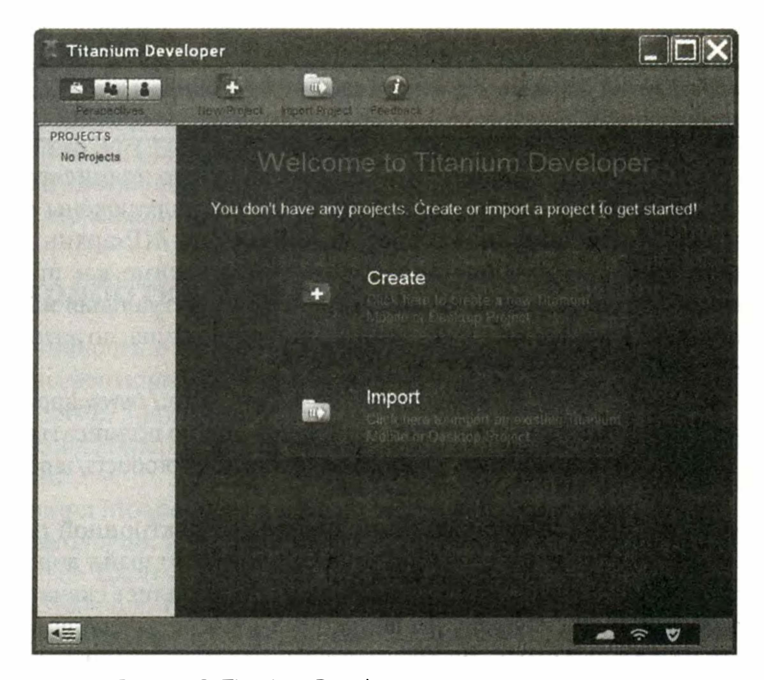


Рис. 17.3. Titanium Developer сразу после установки

ВНИМАНИЕ

При изучении этой главы помните о том, что версии могут различаться. Titanium Developer развивается стремительно. Та версия, которую скачаете вы, вполне возможно, будет отличаться от нашей. В этой главе мы хотим дать вам общее представление о разработке в Titanium. Вам потребуется корректировать материал в зависимости от того, с какой версией Titanium вы работаете. При написании книги мы использовали версию 0.5.0.

В этом разделе мы также хотим познакомить вас с полным объемом функций Titanium Developer IDE. Однако на рис. 17.3 у нас нет еще ни одного проекта, поэтому некоторые меню пока недоступны и мы не можем описать IDE целиком. Покажем скриншот IDE в ситуации, когда один проект у вас уже есть. Таким образом, введение в IDE получится полным. Если в IDE есть хотя бы один проект, она выглядит как на рис. 17.4. Это окно называется **Project Perspective** (Область работы с проектом).

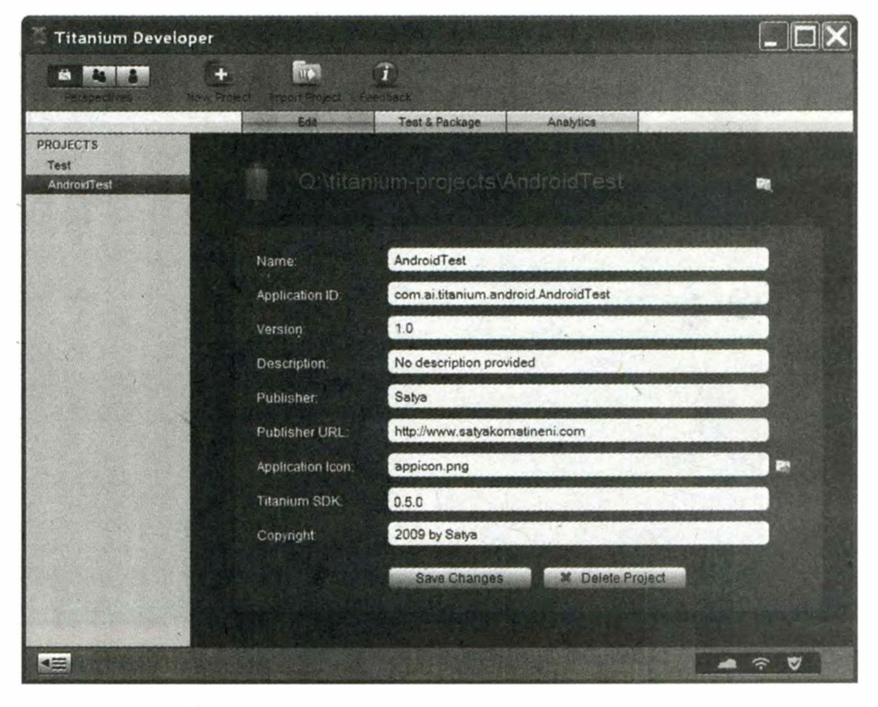


Рис. 17.4. Образец проекта в Titanium Developer

На рис. 17.4 мы имеем два проекта. Их названия указаны слева. Один проект называется `Test`, а другой — `AndroidTest`. `Test` — это программа для ПК, а `AndroidTest` — мобильное приложение для Android. Итак, в одной и той же Titanium Developer IDE можно одновременно разрабатывать и локальные, и мобильные приложения.

Такой пример (или проект), как `AndroidTest`, представляет собой каталог на жестком диске. Например, `AndroidTest`, выделенный на рис. 17.4, находится по адресу `c:\work\AndroidTest`. Другие параметры этого выделенного приложения — просто атрибуты. Один из важнейших атрибутов программы — ID приложения, используемый при создании корневого пакета для файла APK.

ПРИМЕЧАНИЕ

В некоторых версиях Titanium Developer по умолчанию стоит параметр Desktop (Разработка для ПК). В списке параметров вы не найдете Mobile (Разработка для мобильного телефона), так как этот параметр находится не в раскрывающемся списке, а в поле ввода (edit control). Но, если нажать поле ввода, в котором написано Desktop (Разработка для ПК), прямо в нем откроется еще один параметр — Mobile (Разработка для мобильного телефона). Правда, версии Titanium различаются, поэтому здесь возможны нюансы.

Когда проект будет готов к работе, вам придется часто пользоваться функцией Test & Package (Тестировать и создать пакет). Соответствующая вкладка показана на рис. 17.5.

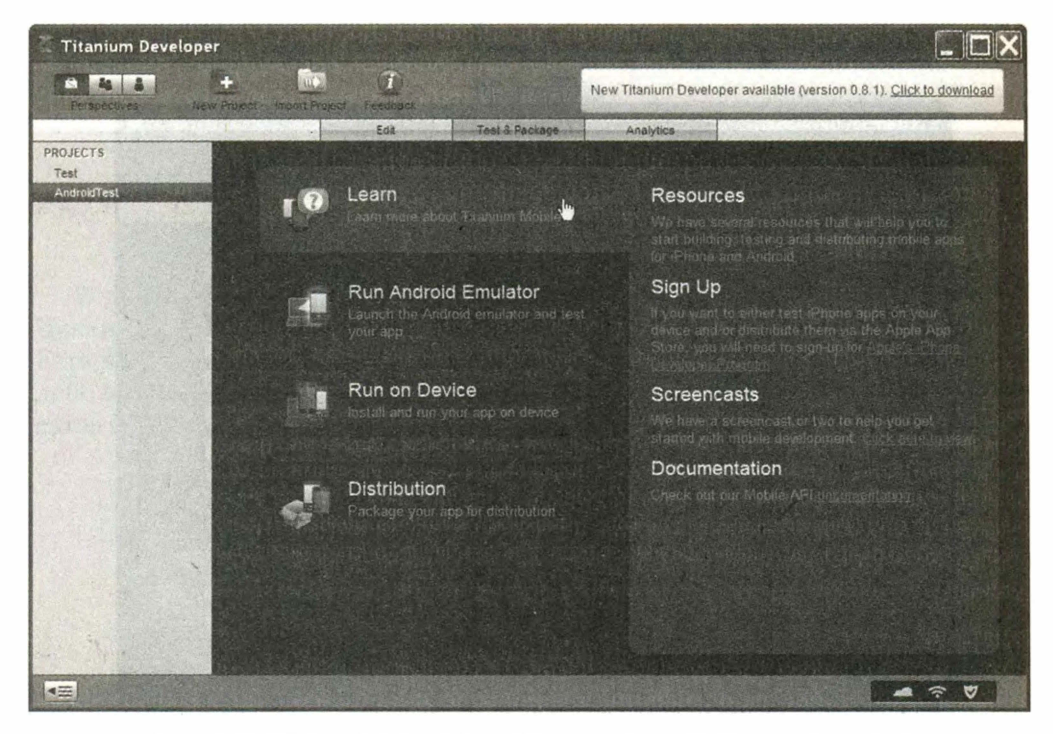


Рис. 17.5. Titanium Developer, вкладка Test & Package (Тестировать и создать пакет)

На этой вкладке берутся ресурсы, необходимые для создания проекта, и развертываются в эмуляторе Android для тестирования. Здесь вы можете работать с программой на устройстве, а также приготовить пакет с приложением Android для распространения. В том числе можно взять приложение и установить его на другой эмулятор, поместить в другую среду разработки либо переместить программу на рынок Android или в хранилище программ Titanium. Некоторые из этих вопросов будут рассмотрены ниже в данной главе.

Еще один толковый элемент Titanium Developer — это хранилище приложений (Application store), в котором можно искать готовые к работе программы. На рис. 17.6 показан соответствующий скриншот. Чтобы попасть в это окно, выберите режим Community (Сообщество) — это средняя кнопка среди трех, расположенных в левом верхнем углу и называемых Perspectives (Режимы). В режиме Community (Сообщество) откройте вкладку Apps (Приложения).

В Titanium Developer также имеется программа-блокнот (scratchpad) (рис. 17.7), в которой можно быстро тестировать образцы кода. Чтобы попасть на этот экран, вернитесь в режим Community (Сообщество) и перейдите на вкладку Sandbox (Песочница).

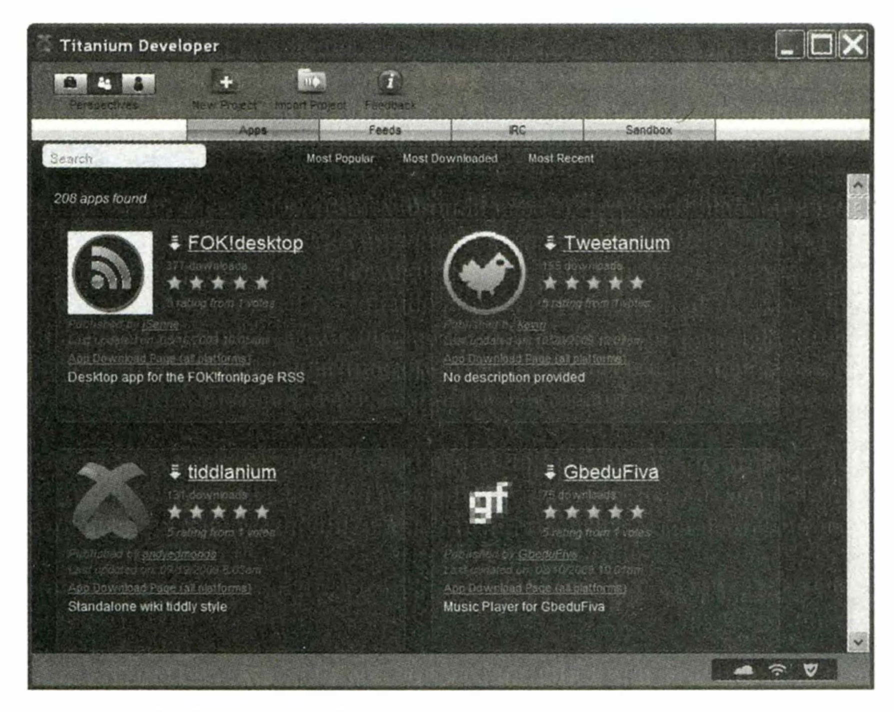


Рис. 17.6. Программа Titanium Developer в хранилище приложений

Из рис. 17.7 сразу становятся понятны несколько вещей. Во-первых, мы видим, что Titanium работает в унисон с некоторыми другими инструментами, в частности jQuery и Mootools, отображаемыми в списке Select JS Libraries.

ПРИМЕЧАНИЕ

При создании нового проекта эти инструменты также доступны как отдельные команды. При выборе таких инструментов Titanium копирует необходимые файлы JavaScript в каталог Resource. Вы также можете сами скачать нужные файлы с официальных сайтов этих продуктов.

В окно запуска, имеющееся в Sandbox (Песочница), можно ввести любой верный код JavaScript или HTML и выполнить его. В качестве примера введите в него следующий фрагмент HTML (листинг 17.2), и нажмите кнопку запуска (см. рис. 17.7).

Листинг 17.2. `Hello World из блокнота Titanium IDE`

```
<html><head></head>
<body>
<h2>Hello World</h2>
</body></html>
```

В листинге 17.3 показан код для фразы `Hello World`, которая будет изображена на экране. Затем в это окно можно добавить следующий скрипт.

Листинг 17.3. `Hello World вместе с JavaScript из блокнота Titanium IDE`

```
<html><head></head>
<body>
```

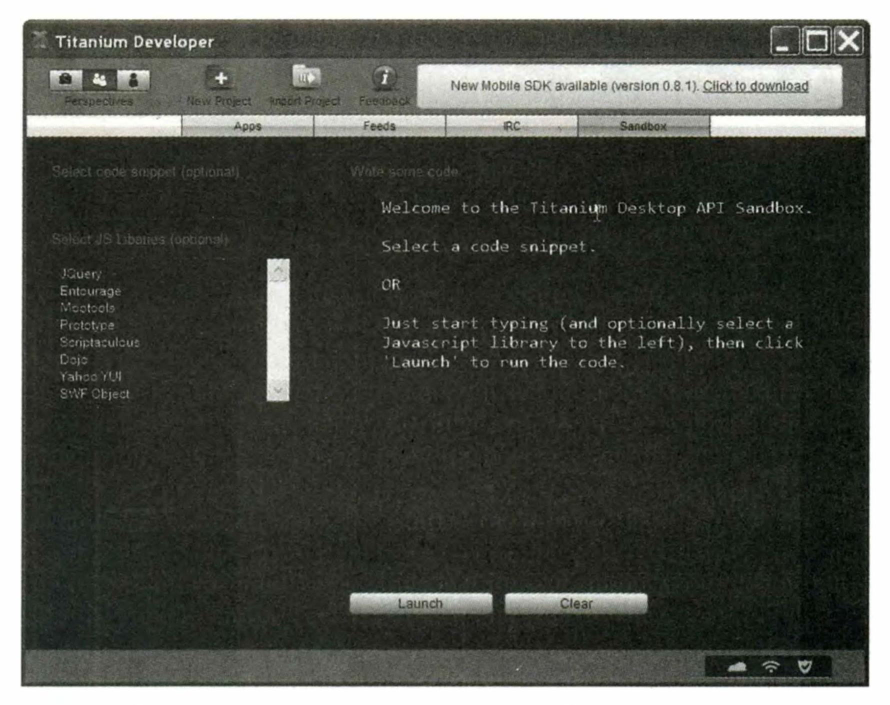


Рис. 17.7. Вкладка Sandbox (Песочница) в Titanium Developer

```
	<h2>Hello World</h2>
<script>
	alert('hello there');
</script>
</body></html>
```

После того как вы запустите этот HTML-файл, на странице с HTML отобразится оповещение JavaScript `hello there`. Эти примеры приведены, чтобы вам была более понятна базовая архитектура, обеспечивающая взаимодействие HTML и JavaScript в Titanium Mobile.

Завершая раздел об установке среды разработки, рассмотрим вкладку с *ретранслируемым интернет-чатом* (IRC), который обеспечивает интерактивную связь с другими разработчиками Titanium, находящимися онлайн (рис. 17.8). Чтобы попасть на этот экран, перейдите в режим Community (Сообщество) (центральная кнопка в верхнем левом углу), а затем откройте вкладку IRC. Нажмите в этом окне кнопку Connect (Соединить), в результате в левой части экрана вы увидите всех разработчиков, которые в данное время находятся онлайн.

Прежде чем перейти к следующему разделу, кратко резюмируем изученный выше материал. Мы подробно рассмотрели архитектуру Titanium Mobile, показали, как этот механизм позволяет программировать на локальном устройстве при помощи технологий, связанных с HTML и JavaScript. Вы научились скачивать и устанавливать Titanium Mobile. Кроме того, мы сообщили вводную информацию о среде разработки Titanium Mobile IDE и ее функциях. Это необходимый багаж знаний для чтения следующего раздела.

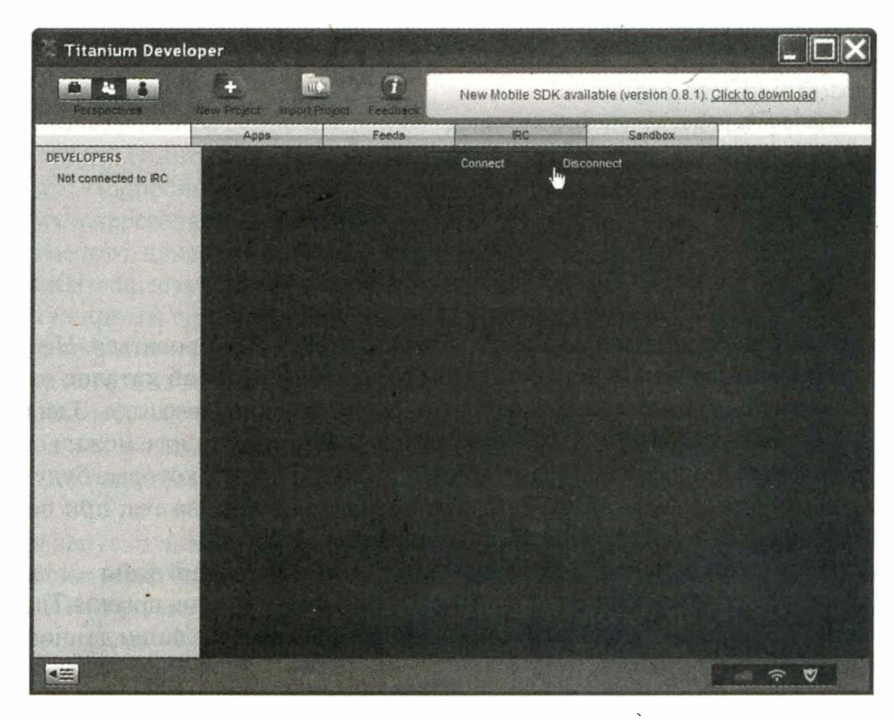


Рис. 17.8. Вкладка IRC вTitanium Developer

Знакомство со связками (ropes): первый проект

Теперь рассмотрим жизненный цикл типового проекта Titanium Mobile. Мы покажем, как создать проект со стандартными параметрами и протестировать его в эмуляторе. Затем мы возьмем этот стандартный проект и изменим, добавив в него собственный HTML-контент, а также отобразим приложение Hello World.

Затем будет рассмотрена пара функций, предназначенных для отладки проекта. Мы научимся запаковывать проект в APK-файл. Наконец, мы научимся подписывать APK-файл и развертывать его в других копиях эмулятора.

Сначала создадим простой проект.

Создание проекта в Titanium Mobile

Чтобы создать в Titanium Mobile новый проект, нажмите пиктограмму New Project (Новый проект) (см. рис. 17.3) в верхней части Titanium Mobile IDE. При создании проекта можно использовать свойства, показанные на рис. 17.4. Проект будем создавать на локальном диске, как это показано в листинге 17.4.

Листинг 17.4. Структура проекта в Titanium Mobile

```
c:\work\AndroidTest1
   \build
```

```
   \android\<eclipse-подобная структура проекта>
\Resources
   \android\appicon.png
      \default.png
   \<здесь находятся наши файлы и подкаталоги>
   \index.html
   \index.css
   \about.html
\manifest
\tiapp.xml
```

В зависимости от версии количество файлов может варьироваться. Но общая структура проекта вполне понятна на этом примере. Основной каталог, как уже было указано в подразделе «Архитектура», — это подкаталог Resources. Здесь нужно создавать файлы HTML, CSS, JavaScript и т. д. В этом каталоге можно создать любое количество других подкаталогов (от Resources и ниже), которые будут нужны для реализации приложения. Этот каталог очень похож на тот, при помощи которого осуществляется администрирование веб-сайта.

За пределами этой директории в корневом каталоге важнейший файл — tiapp.xml. Он представляет собой конфигурационный файл создаваемого нами проекта Titanium. В листинге 17.5 показан файл tiapp.xml, генерируемый при создании данного проекта.

Листинг 17.5. Пример tiapp.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
    <id>com.ai.titanium.android.AndroidTest1</id>
    <name>AndroidTest1</name>
    <version>1.0</version>
    <icon>appicon.png</icon>
   <persistent-wifi>false</persistent-wifi>
   <prerendered-icon>false</prerendered-icon>
   <statusbar-style>opaque</statusbar-style>
   <windows>
      <window>
         <id>initial</id>
         <url>index.html</url>
         <backgroundColor>#111</backgroundColor>
       <icon>ti://featured</icon>
       <barColor>#000</barColor>
       <fullscreen>false</fullscreen>
      </window>
      <window>
         <id>about</id>
         <url>about.html</url>
         <backgroundColor>#111</backgroundColor>
       <icon>ti://top rated</icon>
         <barColor>#000</barColor>
       <fullscreen>false</fullscreen>
      </window>
```

```
  </windows>
</ti:app>
```

Поскольку нашей основной целью является изучение вводного материала по разработке в Titanium, мы не будем подробно разбирать все XML-теги из файла `tiapp.xml`. Подробнее об этих тегах рассказано на сайте сообщества appcelerator: **http://www.appcelerator.com/community/**.

Кроме того, для начала не помешает познакомиться и с этим материалом: **http://www.codestrong.com/timobile/guides/get_started/**.

Полужирным шрифтом в листинге 17.5 выделены узлы, которыми мы собираемся здесь заняться. В теге `window` указано, сколько файлов должно содержаться и отображаться на всех вкладках программы. В листинге 17.5 у нас два окна, одно — для файла `index.html`, а другое — для `about.html`.

Имея эти файлы, автоматически создаваемые в каждом новом проекте, обратимся к вкладке **Test & Package** (Тестировать и создать пакет) Titanium Developer (см. рис. 17.5) и выберем запуск проекта в эмуляторе.

При запуске новое приложение будет выглядеть как на рис. 17.9. Программа должна запуститься в эмуляторе автоматически, без вашего участия, точно как Eclipse ADT. Если это почему-то не происходит — перейдите в меню Android и запустите ее сами.

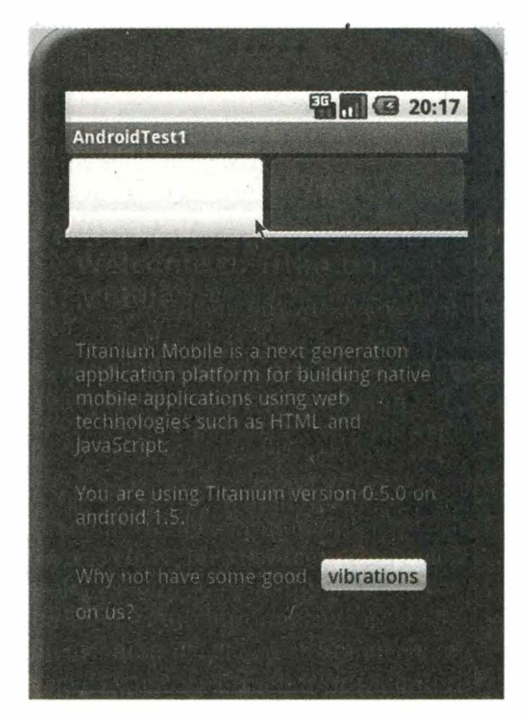


Рис. 17.9. Пример многооконного приложения в Titanium Developer

Обратите внимание на `index.html` и `about.html`: как видите, `index.html` находится на первой вкладке.

Осваиваем Hello World

Теперь посмотрим, как упростить наш первичный проект — разберемся с вкладками и сделаем все в одном окне. Кроме того, изменим цвет фона на белый, чтобы первая страница выглядела как на рис. 17.10.

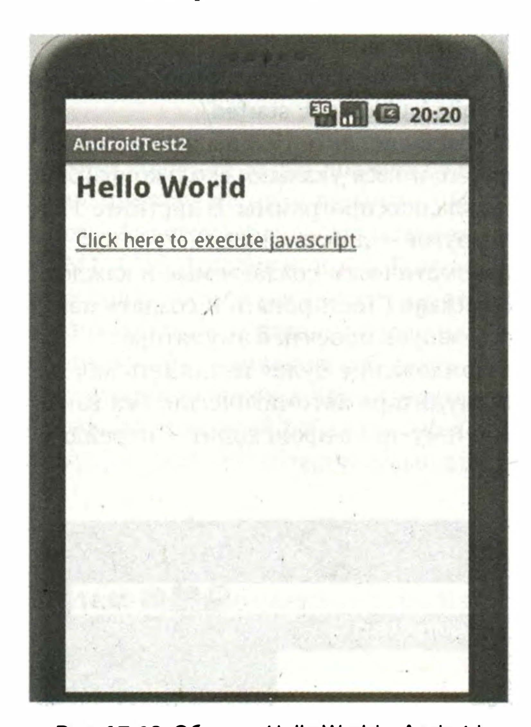


Рис. 17.10. Образец Hello World в Android

Для этого нужно будет сделать следующее.

1. Приготовить новый index.html или изменить имеющийся, чтобы страница выглядела как на рис. 17.10.
2. Изменить файл tiapp.xml так, чтобы осталось только одно окно, а URL этого окна указывала на файл index.html. Кроме того, изменить цвет фона окна на более светлый.

Сейчас выполним оба этих этапа.

Файл index.html, необходимый для создания этой страницы, показан в листинге 17.6. Можете заменить index.html, автоматически созданный IDE, этим файлом.

Листинг 17.6. Самостоятельный пример Hello World

```
<html><body>
<h2>Hello World</h2>

<p><a href="javascript:alert('hello')">
Click here to execute JavaScript
</a></p>
</body></html>
```

Этот файл index.html довольно прост. Страница предназначена для вызова функции JavaScript alert, приветствующей вас при нажатии фразы **Click here to execute JavaScript** (Нажмите, чтобы выполнить JavaScript).

Имея такой модифицированный файл index.html, посмотрим, что нужно изменить в tiapp.xml. Ниже приведен обновленный файл tiapp.xml (листинг 17.7).

Листинг 17.7. tiapp.xml с одним окном

```
<?xml version="1.0" encoding="UTF-8"?>
<ti:app xmlns:ti="http://ti.appcelerator.org">
    <id>com.ai.titanium.android.AndroidTest2</id>
    <name>AndroidTest</name>
    <version>1.0</version>
    <icon>appicon.png</icon>
   <persistent-wifi>false</persistent-wifi>
   <prerendered-icon>false</prerendered-icon>
   <statusbar-style>opaque</statusbar-style>
    <windows>
      <window>
         <id>initial</id>
         <url>index.html</url>
         <backgroundColor>white</backgroundColor>
       <icon>ti://featured</icon>
       <barColor>#000</barColor>
       <fullscreen>false</fullscreen>
      </window>
    </windows>
</ti:app>
```

Важные элементы выделены полужирным шрифтом. Обратите внимание — теперь на index.html указывает только одно окно, а фон стал белым.

ПРИМЕЧАНИЕ

Обратите внимание на то, что в наших примерах мы присваиваем приложениям порядковые номера: AndroidTest1, AndroidTest2 и т. д. Это делать не обязательно. Мы поступаем так, поскольку нам проще получать нужные скриншоты. Иначе, изменив код, мы потеряем предыдущий тестовый экземпляр. Надеемся, что это маленькое неудобство не мешает вам следить за процессом.

Имея файлы index.html и tiapp.xml, можно запаковать и протестировать это приложение в эмуляторе. Теперь, запуская эту программу в Titanium Mobile IDE, в эмуляторе вы увидите такое же изображение, как на рис. 17.10.

Подготовка приложения к отладке

Одна из причин, по которой мы включили в наш пример с index.html оповещение JavaScript (см. листинг 17.6), — необходимость протестировать JavaScript и посмотреть, как он работает. Оповещение для этого как раз подходит. На практике многие программисты используют оповещения JavaScript как отладочный инструмент.

Однако, как вы увидите, JavaScript из index.html (см. листинг 17.6) не будет выводить приветствие при запуске в эмуляторе. А при запуске на ПК — будет. В чем же дело?

По-видимому, специалисты Appcelerator Inc. переопределили эту функцию для записи сообщения во внутреннюю отладочную консоль. Оказывается, WebKit, являющийся хостом для веб-страницы (в данном случае — index.html) позволяет клиенту точно задать значение оповещения. На платформе Android компания Appcelerator решила перенаправить эту информацию в файл регистрации, а не выводить на консоль. Зачем — остается только догадываться.

В принципе, этому может быть два объяснения. В сфере JavaScript разработчики все активнее пользуются оповещениями именно в целях отладки. Возникает аргумент: если alert служит в первую очередь для отладки, почему бы просто не записывать сообщение в файле регистрации, а на экран не выводить? Такое бесполезное сообщение будет только отвлекать от работы с программой. Вторая причина заключается в самой природе Android. Ведь диалоговые окна в Android являются асинхронными. Поэтому мы делали бы слишком большой «крюк», заставляя диалоговое окно ждать — а так обычно и происходит в JavaScript.

Есть два способа обойти эту проблему. Первый — использовать API пользовательского интерфейса Titanium и создать нативное окно уведомления Android с кнопкой OK. Второй — воспользоваться отладочным API Titanium. Ниже рассмотрим оба варианта.

В листинге 17.8 показан index.html, переписанный с учетом каждого из этих вариантов.

Листинг 17.8. index.html, использующий альтернативные отладочные параметры

```
<html><head>
<script>
function myalert(message)
{
   var a = Titanium.UI.createAlertDialog();
   a.setMessage(message);
   a.setTitle('My Alert');
   a.setButtonNames(["OK"]);
   a.show();
}

function dalert(message)
{
   Titanium.API.info(message);
   alert(message);
   // var a = prompt(message);
   myalert(message);
}
</script>

</head>

<body>
<h2>Hello World</h2>

<p><a href="javascript:dalert('hello')">
```

```
Click here to execute JavaScript
</a></p>

</body></html>
```

В функции myalert используется диалоговая составляющая Android. При вызове этой функции экран будет выглядеть как на рис. 17.11. Этот способ удобен, но не следует забывать, что нативные диалоговые окна Android являются асинхронными и для более эффективной работы они должны быть с подсчетом ссылок (reference-counted). Возможно, вы решите пользоваться ими экономно.

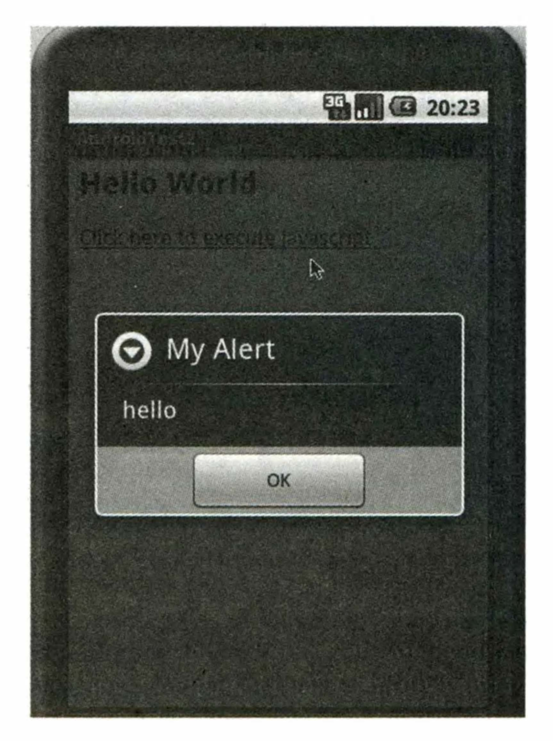


Рис. 17.11. Нативное окно оповещения Titanium

Второй вариант обойти указанную выше проблему — использовать отладочный API Titanium для записи сообщений об отладке вместо оповещений. Так работает функция dalert. Эти сообщения идут в окно консоли Titanium Mobile, которое отображается, когда вы работаете с вкладкой запуска. Для активации вкладки запуска необходимо сделать следующее.

1. Выбрать имя проекта, щелкнув на нем в списке.
2. Щелкнуть на вкладке Test & Package (Тестирование и упаковка).
3. Нажать вертикальную вкладку эмулятора Android. Тогда окно консоли отобразится справа, внутри IDE; в его нижней части будут располагаться две кнопки: Launch (Пуск) и Stop emulator (Остановить эмулятор).

4. Для запуска программы нажмите Launch App (Пуск приложения). В ходе работы программы отладочные сообщения Titanium будут отображаться в расположенном справа окне консоли.

   На рис. 17.12 показан соответствующий пример.

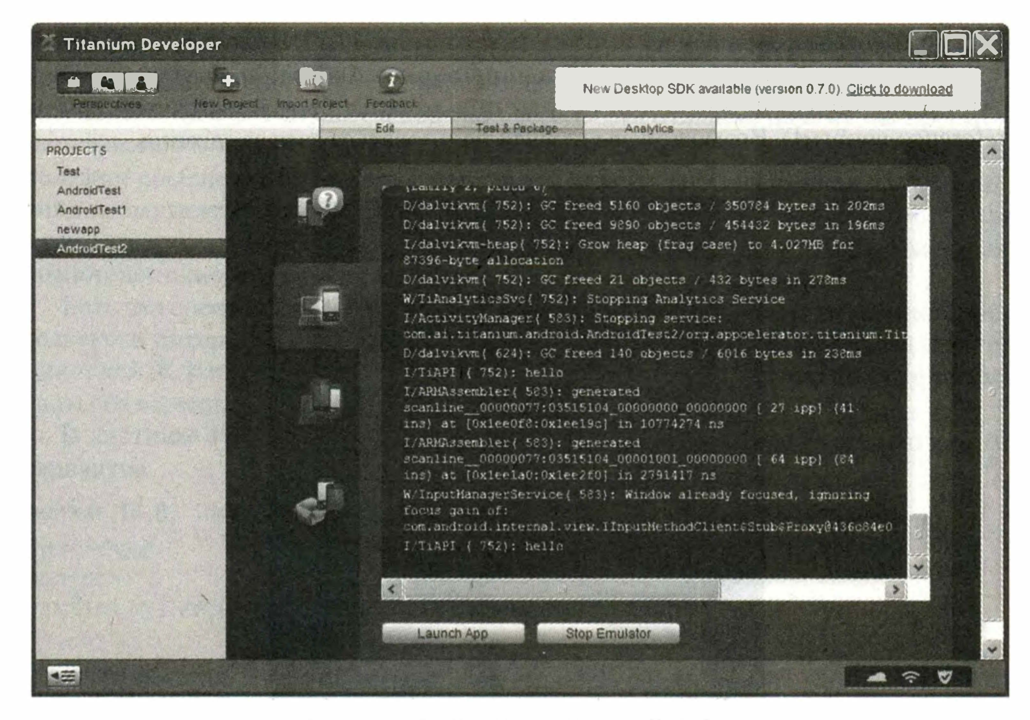


Рис. 17.12. Окно консоли в Titanium Mobile

Обратите внимание на строку в центре экрана: `I/TiApi: hello`. Это сообщение было отправлено при помощи метода `Titanium.API.info(message)`. (Кстати, функция-«подсказка» в JavaScript работает прекрасно.) Если вы пытаетесь сделать это впервые, то не забывайте, что может случиться такой «сюрприз».

Теперь, когда у нас есть программа и мы знаем, как производится ее отладка, запакуем ее в APK-файл.

Упаковка приложения

В Android модулем развертывания (unit of deployment) является файл пакета Android или APK (работа с APK-файлами подробно рассмотрена в главе 7). Чтобы экспортировать куда-либо вашу программу, вам понадобится такой APK-файл.

В Android требуется подписать APK-файл, чтобы его можно было развертывать на эмуляторе, устройстве или электронном рынке. На самом деле при разработке и тестировании программ Android в Eclipse ADT ADT подписывает их (в фоновом режиме) встроенным ключом, пригодным только для развертывания программы в эмуляторе. Кроме того, Android работает с APK-файлами, установленными с оди-

наковой подписью, немного специфически, так как они совместно используют пространство процессов.

ПРИМЕЧАНИЕ

Если два APK-файла совместно используют пространство процессов, то при этом они также совместно применяют создающую его виртуальную машину Java. Они также могут совместно использовать общие переменные, но, если в одном APK-файле возникнут проблемы, они могут затронуть и другой APK-файл.

Совместное использование может быть и злом, и благом, в зависимости от того, какая степень изоляции вам требуется. Подпись также важна для установки программных обновлений.

Подписывание программ более подробно рассмотрено в главе 7. Кроме того, можете почитать об этом здесь: http://developer.android.com/guide/publishing/app-signing.html.

В качестве ремарки: Android создает хранилища ключей, используемых при разработке или отладке, в следующих местах:

- в Mac OS X и Linux — ~/.android/debug.keystore;
- Windows XP — C:\Documents and Settings\\.android\debug.keystore;
- Windows Vista — C:\Users\\.android\debug.keystore.

Чтобы самостоятельно подписать APK-файл, необходимо понимать внутреннюю структуру ключа, существующую в JDK (инструментарий для разработки в Java) (В главе 2 было сказано, что для запуска Eclipse нужно скачать совместимый JDK.) Для создания хранилища ключей с паролем используется инструмент keytool из JDK (подробнее об этом — в главе 7 или по адресу http://java.sun.com/javase/6/docs/technotes/tools/windows/keytool.html). В листинге 17.9 показано, как создается хранилище ключей с ключом mykey по адресу /jre/bin/keytool.

Листинг 17.9. Параметры keytool

```
keytool
   -genkey // генерирование открытого/закрытого ключа
   -alias mykey // имя ключа
   -keystore c:\somekeystore.store // расположение хранилища
   -storepass abc // пароль
   -keypass abc // пароль
   -keyalg RSA  // алгоритм ключа
   -validaty 14000 // срок годности ключа (в днях)
```

Создав хранилище ключей, можно использовать вкладку **Distribution** (Распространение), чтобы создать APK-файл. Titanium Developer подскажет имя ключа, которым нужно будет подписать пакет. Необходимо знать путь к хранилищу ключей и пароль к этому хранилищу. Если вы не вспомните имя хранилища (это имя также называется alias — «псевдоним»), то для перечисления записей из хранилища ключей можно воспользоваться инструментом keytool. Но вам обязательно нужно будет записать путь к хранилищу ключей и пароль от него и сохранить эту информацию где-нибудь в надежном месте. Вот пример команды, которая выводит список записей из хранилища ключей:

```
keytool -list c:\somekeystore.store
```

В фоновом режиме Titanium применяет jarsigner (рассмотренный в главе 7) для взятия указанного вами псевдонима ключа и подписывания APK-файла. Затем Titanium создает сам APK-файл в указанном вами каталоге.

Здесь необходимо отметить, что Eclipse ADT (также рассмотренный в главе 7) позволяет создавать APK-файл без подписи (затем вы сами подписываете его при помощи keytool и jarsigner), но Titaninum Developer работает только с подписанными APK-файлами, чтобы их можно было использовать и за пределами среды разработки.

Для тестирования программы в эмуляторе и Eclipse AST, и Titanium IDE подписывают APK-файлы встроенным ключом, который подходит только для пробного развертывания в эмуляторе. Мы уже указывали выше, где эти разработочные ключи хранятся в Eclipse ADT. В Titanium IDE есть похожее хранилище. Однако эта информация на практике бывает нужна нечасто.

Установка APK-файла на собственном эмуляторе

Имея в распоряжении APK-файл, можно продолжить установку этого файла на том эмуляторе Android, которым вы обычно пользуетесь (когда не работаете с Titanium) для тестирования других программ Android. Чтобы начать работу, нужно запустить эмулятор следующим образом:

```
\android\tools\emulator @avdname
```

Здесь avdname — это название виртуального устройства Android (AVD). В главе 2 рассказывается, что такое AVD и как их создавать. Это механизм, позволяющий одновременно работать с несколькими эмуляторами, каждый из которых занимает отдельный уровень в Android SDK, а также предоставляющий инструментарий для тестирования. Создать AVD не так просто, но для этого есть специальная быстрая команда:

```
android create avd -t 3 -c 32M -p ..\avds\avd3 -n avd 3
```

Здесь -t — это нужный уровень Android SDK, -c — память, -p — путь, а avd3 — название виртуального устройства.

Когда эмулятор уже запущен и работает, можно использовать команды из листинга 17.10 для установки и деинсталляции пакета.

Листинг 17.10. Параметры для установки и деинсталляции adb

```
adb install [-l] [-r]
- отправить этот пакет на устройство и установить его
   ('-l' означает блокировку пересылки (forward-lock) в программе)
   ('-r' означает переустановку приложения с сохранением его данных)

adb uninstall [-k]
- удаление данного пакета приложения с устройства
   ('-k' означает сохранение данных и директории кэша)
```

Обратите внимание: при установке используется имя файла, а при деинсталляции — полное Java-подобное имя пакета. Что касается нашей тестовой программы AndroidTest2, ее полное Java-подобное имя пакета будет таким:

```
com.ai.android.titanium.AndroidTest2
```

Кроме того, непосредственно при помощи эмулятора или устройства можно деинсталлировать любое приложение. Для этого нужно сделать следующее.

1. Запустить эмулятор.
2. В меню выбрать пункт Dev Tools (инструменты разработки).
3. Перейти в Package browser (Обозреватель пакетов) и найти наш пакет.
4. Пока имя пакета выделено, выбрать меню.
5. В меню выбрать Delete Package (Удалить пакет).

Для запуска эмулятора нам вновь понадобится имя avd. Здесь пригодятся две следующие команды. Первая строит список имеющихся виртуальных устройств Android, а вторая запускает в эмуляторе указанное avd:

```
\tools\android list avd // строит список имеющихся avd
\tools\emulator @avdname // запускает указанное avd
```

Разумеется, сделав все это, мы наконец сможем запустить в эмуляторе любое выбранное приложение. После запуска вы увидите на экране эмулятора картину, показанную на рис. 17.10.

В заключение этого раздела опишем еще одну интересную функцию, доступную в проекте Titanium. На этапе сборки (build) Titanium создает директорию android в подкаталоге build. По существу, эта директория содержит полнофункциональный проект Eclipse ADT. В Windows все, что требуется сделать, — переместить файл tiapp.xml и подкаталог Resources в build/android/assets. Тогда подкаталог build/android будет полным, и вы сможете продолжить тестирование или разработку в нативной среде Android, но уже под Eclipse ADT. Если возникнут ошибки сборки, R.java можно удалить.

Планирование приложений для практического использования

До сих пор в этой главе мы подразумевали, что, вооружившись HTML или JavaScript, мы сможем писать любые приложения. Но насколько это справедливо на практике? В любом приложении вам понадобятся механизмы для программирования гибких пользовательских интерфейсов с формами, медиа, данными и т. д. Будет необходим механизм для реализации бизнес-логики, а также элементы для считывания информации из базы данных и сохранения состояний.

Сначала поговорим об относительно простых моментах: промежуточном программном обеспечении (middleware) и данных. JavaScript, благодаря использованию Ajax и JSON, в любое время может запрашивать данные при помощи серверов. Серверы в данном случае действуют как каналы для полного или частичного сохранения бизнес-логики. Если программе потребуется доступ к локальным ресурсам или базам данных, для этого всегда можно использовать SQLite. Но следует ожидать, что в некоторых особых случаях лучше будет использовать облачные сервисы и трехзвенные модели (three-tier model). Такой подход не должен вызывать особых проблем.

Однако фреймворк пользовательского интерфейса — это совсем другое дело. Такие фреймворки, как Swing и WPF (Windows Presentation Framework), очень

сложны. Насколько обычный HTML и JavaScript способны решать задачи, возникающие при создании пользовательского интерфейса?

Хотя некоторые сложности и останутся в любом случае, такой инструмент, как jQuery (его основа — JavaScript) предоставляет мощную модель для динамического изменения дерева объектной модели документа в HTML и позволяет решать некоторые возникающие вопросы.

Библиотека jQuery легка, проста и расширяема. В названии jQuery есть слово query, то есть «запрос». Это свидетельствует о ее способности запрашивать любой узел HTML DOM посредством сжатого синтаксиса, в частности с применением селекторов CSS и выражений XSLT. Итак, запрос jQuery по своей природе гораздо лучше подходит для работы с пользовательскими интерфейсами и с HTML, а не с базами данных. В этой главе будет приведено несколько примеров, демонстрирующих данный механизм на практике.

JQuery — всего лишь один из длинного ряда инструментов, которые используют JavaScript и HTML для полномасштабного программирования пользовательских интерфейсов. Можете сами исследовать альтернативные варианты и выбрать тот, который подойдет вам лучше всего.

А теперь приглашаем вас на небольшую экскурсию по jQuery.

Базовое руководство по jQuery

Домашняя страница jQuery расположена по адресу **http://jquery.com/**. Отсюда вы сможете одним файлом (на JavaScript) скачать всю библиотеку. Это примерно 100 Кбайт кода. После скачивания вы сможете вставлять этот код в ваши HTML-страницы, пользуясь сегментом, приведенным в листинге 17.11.

Листинг 17.11. Включение jQuery в HTML-файл

```
<script src="../../js/jquery132-dev.js"></script>
```

В целом эта библиотека документирована очень хорошо. Поэтому любой может быстро и без лишних усилий научиться с ней работать. Вы сможете с успехом воспользоваться этими знаниями и при серверном HTML-программировании.

Посмотрим, что мы можем в данном случае предпринять. Одна из вещей, которая может понадобиться в HTML, — нахождение тега `<div>` или абзаца и замена содержимого этого элемента определенным текстом. Кроме того, можно изменить стиль `<div>` или скрыть этот `<div>`. Давайте рассмотрим это подробнее.

Листинг 17.12. Примеры выборки при помощи jQuery

```
function replaceAParagraph(newText)
{
   // находим HTML-элемент по его ID
   // возвращается массив подходящих элементов
   var myParagraph = $("#MyParagraphID")[0];

   // считывание имеющегося HTML из элемента
   var oldText = myParagraph.html();

   // замена его новым
```

```
    myParagraph.html(newText);

    // или более простой формат
    $("#MyParagraphID").html(newText);

    // изменяем стиль этого элемента
    $("#MyParagraphID").css("color:red;");

    // скрываем элемент
    $("#MyParagraphID").hide();
}
```

$ — это функция, входящая в состав jQuery и использующая селекторы для нахождения нужного элемента. Ее синтаксис для нахождения элемента тщательно проработан и занимает центральное место в jQuery. В листинге 17.13 показан один из многих способов применения селекторов.

Листинг 17.13. Различные селекторы jQuery

```
$("#MyElementID") // конкретный id
$(".MyClass") // все элементы, относящиеся к данному классу
$("p") // все параграфы
$("p.MyClass") // параграфы с MyClass
$("div") // все div
$(".MyClass1.MyClass2.MyClass3") // нахождение трех классов
$("div,p,p.MyClass,#MyElementID") // совпадение со всеми этими классами

// элементы-потомки первого уровня
$("#Main > *") // все потомки Main
$("parent > child")

// потомки первого и второго уровней
$("ancestor descendents")
$("form input") // все поля ввода в форме

$("label + input") // все поля ввода после обозначения
$("prev + next")

// начиная с myclass, находим соседние смежные элементы типа div
$(".myclass ~ div")
$("prev ~ next)
```

Когда при помощи таких селекторов будет выбран определенный ряд элементов, полученные в итоге исходящие узлы (output nodes) можно отфильтровать при помощи следующего синтаксиса:

```
$("selector:criteria")
```

Вот как используется этот селектор и синтаксис критериев:

```
$("tr:even").css("background-color", "#bbbbff");
```

В данном примере выбираются все четные стоки таблицы, а затем для них задается стиль. Некоторые из возможных критериев приведены в листинге 17.14.

Листинг 17.14. Критерии выборки в jQuery

```
first
last
even
odd
eq(index)
lt(index)
gt(index)
header   //(h1, h2 и т. д.)
animated
```

В листинге 17.15 приведено еще несколько примеров, взятых из документации по jQuery в немного измененном формате.

Листинг 17.15. Операция наведения указателя на абзац

```
function hoverParagraph()
{
   $("p").hover(function () {
         $(this).css({'background-color' : 'yellow',
                   'font-weight' : 'bolder'});
      }, function () {
         var cssObj = {
          'background-color'   '#ddd',
          'font-weight' : '',
          'color' : 'rgb(0,40,244)'
         }
         $(this).css(cssObj);
      });
}
```

В этом примере мы нашли абзац и зарегистрировали набор функций обратных вызовов, выполняемых при наведении указателя на этот абзац. Первая функция изменяет CSS абзаца — фон становится желтым, а шрифт — полужирным. Вторая функция изменяет этот стиль на другой, когда мы убираем указатель.

В листинге 17.16 показана ситуация, в которой указатель уходит с текста.

Листинг 17.16. Работа с отведением указателя

```
function paragraphMouseover()
{
   $("p").mouseover(function () {
      $(this).css("color","red");
   });
}
```

В этом примере CSS абзаца изменяются при помощи анонимной функции в случае наведения на абзац указателя мыши.

Базовое руководство по продвинутому JavaScript

Привыкая к JavaScript-центричным технологиям, таким как jQuery и Titanium (в котором используются API JavaScript), вы начнете замечать приемы работы,

распространенные в JavaScript, но достаточно непривычные для тех, кто работает с этим языком лишь от случая к случаю при дополнении веб-страниц.

Первый сюрприз — это равнозначность массива и объекта. Позвольте разоблачить это волшебство. Начнем с объявления или инициализации объекта в JavaScript:

```
var myobj = {};
```

Здесь фигурные скобки указывают начало и конец инициализации объекта. В данном примере в фигурных скобках ничего нет. Так мы сообщаем JavaScript, что myobj — это объект, не содержащий никакого содержимого или членов. Но объект при этом все же определяется. Расширим этот шаблон инициализации (initialization pattern):

```
var myobj = {name:"phone-number1",value:"123456"};
```

Эта инструкция позволяет вам сделать следующее:

```
alert(myobj.name());
or
alert(myobj["name"]);
```

Так проверяется равнозначность объектов и массивов ассоциативных элементов и демонстрируется, что внутри системы члены объекта представлены как ассоциативный массив. Верно также обратное утверждение.

```
var myobj={};
myobj["name"] = "aaaa";
myobj["value"] = "bbbb";
```

Следующие инструкции также будут идентичны.

```
alert(myobj.name());
or
alert(myobj["name"]);
```

Такой способ инициализации объектов очень удобен. Рассмотрим фрагмент кода, которым будем пользоваться в следующем разделе (листинг 17.17).

Листинг 17.17. Пример определения массива в JavaScript

```
var itemArray = [
   {name: "Social", value: "12345678"},
   {name: "cell1", value: "12345678"},
   {name: "cell2", value: "12345678"}
];
```

Так можно быстро определить массив, состоящий из трех объектов, в каждом из которых содержится пара «имя/значение». Шаблон инициализации объекта также допускает присутствие вложенных объектов. В листинге 17.18 приведен пример.

Листинг 17.18. Инициализация вложенных объектов

```
var someobj = {field1:10,
field2:"string",
field3:{field1:10,field2:"string"}};
```

В принципе, здесь мы видим основы JSON ((JavaScript Object Notation). Формат JSON часто используется для передачи данных между клиентом и сервером. Если эта концепция вам не знакома, можете почитать о JSON (http://json.org), так как вам без него не обойтись при обмене информацией с веб-сервером, нахождении и сохранении данных с использованием протокола HTTP.

Теперь коротко рассмотрим анонимные функции. В листинге 17.19 приведен пример.

Листинг 17.19. Анонимные функции

```
function Person() {
   var age = 40;   //начальное значение
   this.setAge = function(howold) { age = howold };
   this.firstname = "First";
   this.lastname = "Last";
}
var me = new Person();
me.firstname = "aaaa";
me.lastname = "bbbb";
me.setAge(25);
// вот это — неправильно
me.age=44;
```

В предыдущем примере член setAge определен как анонимная функция, имеющая доступ к индивидуальной переменной age, а firstname и lastname здесь — общие переменные.

В заключение этого раздела поговорим о пространствах имен JavaScript, так как они очень активно используются в библиотеках, базирующихся на JavaScript.

Рассмотрим следующий пример (листинг 17.20), который часто приводится в качестве иллюстрации работы с пространствами имен в JavaScript.

Листинг 17.20. Пространства имен в JavaScript

```
var MY_NAME_SPACE = function() {
   return {
      method_1 : function() {
         // здесь выполняются операции
      },
      method_2 : function() {
         // здесь выполняются операции
      }
   };
}();
```

На этом примере несложно понять, как осуществляется кодирование в JavaScript. В упрощенном виде приведенный выше код имеет следующую форму:

```
MY_NAME_SPACE.method_1();
MY_NAME_SPACE.method_2();
```

Префикс MY_NAME_SPACE исключает конфликт с другими библиотеками. Разберемся, что происходит. Начнем с оператора возврата (return statement). Если вы встретите шаблон:

```
var someobj = {method_1: function() {}, method_2: function2() {}}
```

это значит, что далее произойдет инициализация объекта с двумя членами, method_1 и method_2, которые являются анонимными функциями. Итак, если мы имеем дело с объектом, то у приведенной выше инструкции такой вид:

```
var MY_NAME_SPACE = function() { return someobj; }();
```

где someobj — это объект с функциями method1 и method2. Поэтому, если за MY_NAME_SPACE не будут следовать скобки (), это будет функция, а не объект — объектом является someobj, содержащий методы. А нам нужно, чтобы с someobj можно было произвести операцию вида someobj.method1(). В принципе, в скобках и выполняется анонимная функция, заставляющая MY_NAME_SPACE указывать на someobj, возвращаемый в результате. Вот как это можно сделать:

```
MY_NAME_SPACE.method_1();
MY_NAME_SPACE.method_2();
```

Иногда этот шаблон будет записываться так:

```
var MY_NAME_SPACE = (function() {
 return {
     method_1 : function() {
         // здесь выполняются операции
     },
     method_2 : function() {
         // здесь выполняются операции
     }
 };
})();
```

Понятие о механизме для создания микрошаблонов

Когда мы начинаем работать с HTML как с фреймворком для пользовательских интерфейсов, вскоре становится понятно, насколько полезен может быть шаблонизатор (templating engine), наподобие тех, которые применяются в технологиях JSP или ASP. Например, рассмотрим следующий массив JavaScript:

```
var itemAarray = [
   {name: "Social", value: "12345678"},
   {name: "cell1", value: "12345678"},
   {name: "cell2", value: "12345678"}
];
```

Допустим, нам нужно создать HTML-страницу, которая будет выглядеть вот так:

```
Social
12345678
Cell1
12345678
Cell2
12345678
```

Создавать все эти HTML-узлы на лету очень непросто, даже вооружившись jQuery. Но мы можем воспользоваться шаблоном, который выглядит как обычная страница JSP (листинг 17.21).

Листинг 17.21. Пример HTML-шаблона

```
<#
for(var i=0; i < itemArrayData.length; i++)
{
   var item = itemArrayData[i];
#>
   <p><#=item.name #>:<#=item.value #></p>
<# } #>
```

Здесь комбинация символов <%=, применяемая в JSP, заменяется на <#. Чтобы дополнить этот шаблон приведенным выше набором данных JavaScript, нам нужен определенный шаблонизатор, который можно будет выполнять на JavaScript. Джон Резиг (John Resig), автор библиотеки jQuery, написал такой шаблонизатор. Благодаря своей «миниатюрности» этот движок стал называться The JavaScript Microtemplating Engine (движок микрошаблонов JavaScript).

В листинге 17.22 приведен весь исходный код (вариант Джона Резига, впоследствии дополнявшийся в веб рядом специалистов), который можно сохранить в отдельном файле для последующего включения в код.

Листинг 17.22. Написанный Джоном Резигом код движка для создания микрошаблонов

```
var _tmplCache = {}
this.parseTemplate = function(str, data) {
   /// <аннотация>
   /// Клиентский синтаксический анализатор шаблона,
   /// использующий в коде выражения <#= #>
   /// и <# code #>.
   /// А также блоки кода # # для расширения шаблона.
   /// ПРИМЕЧАНИЕ: в некоторых ситуациях может запинаться
   /// на одинарных кавычках.
   /// При применении в тексте литералов используйте '
   /// и всячески избегайте одинарных кавычек как
   /// разделительных знаков между атрибутами.
   /// </summary>
   /// <param name="str" type="string">текст шаблона,
   /// который нужно расширить</param>
   /// <param name="data" type="var">
   /// Любые данные для слияния. Передайте объект,
   /// и свойства этого объекта будут видны как переменные.
   /// </param>
   /// <returns type="string" />
   var err = "";
   try {
      var func = _tmplCache[str];
      if (!func) {
         var strFunc =
         "var p=[],print=function(){p.push.apply(p,arguments);};" +
```

```
                "with(obj){p.push('" +
//                 str
//         .replace(/[\r\t\n]/g, " ")
//         .split("<#").join("\t")
//         .replace(/((^|#>)[^\t]*)'/g, "$1\r")
//         .replace(/\t=(.*?)#>/g, "',$1,'")
//         .split("\t").join("');")
//         .split("#>").join("p.push('")
//         .split("\r").join("\\'") + "');}return p.join('');";

        str.replace(/[\r\t\n]/g, " ")
           .replace(/'(?=[^#]*#>)/g, "\t")
           .split("'").join("\\'")
           .split("\t").join("'")
           .replace(/<#=(.+?)#>/g, "',$1,'")
           .split("<#").join("');")
           .split("#>").join("p.push('")
           + "');}return p.join('');";

        // alert(strFunc);
        func = new Function("obj", strFunc);
        _tmplCache[str] = func;
     }
     return func(data);
   } catch (e) { err = e.message; }
   return "< # ERROR: " + err.htmlEncode() + " # >";
}
```

Теперь рассмотрим файл `index.html`, в котором используются рассмотренные выше концепции, а также берется массив объектов и генерируется HTML-представление (листинг 17.23). Потребуется либо скачать и включить в проект файл `jquery.js` с сайта jQuery, либо использовать файл, поставляемый с Titanium. Если вы собираетесь работать с тем файлом, который входит в состав Titanium, то нужно выбрать его из списка инструментов при создании проекта (нам показалось, что проще скачать файл с сайта jQuery). Если при создании проекта вы пропустили этот шаг, то воспользоваться этим вариантом позже вы не сможете. Кроме того, нужно создать файл `template-engine.js` при помощи механизма микрошаблонов, взяв приведенный выше код и вставив его в соответствующий подкаталог ниже `Resources`.

Листинг 17.23. HTML, использующий механизм для создания микрошаблонов

```
<html><head>
<script src="../../js/jquery132-dev.js"></script>
<script src="../../js/template-engine.js"></script>

<script>
// данные
var itemArray = [
          {name: "Social", value: "12345678"},
          {name: "cell1", value: "12345678"},
```

```
            {name: "cell2", value: "12345678"}
            ];

function onloadFunction()
{
   var s = $("#MyTemplate").html();
   var s1 = parseTemplate(s, {itemArrayData: itemArray});
   $("#target").html(s1);
}
</script>

<script id="MyTemplate" type="text/html">
 <#
 for(var i=0; i < itemArrayData.length; i++)
 {
   var item = itemArrayData[i];
 #>
   <p><#=item.name #>:<#=item.value #></p>
 <# } #>
</script>

</head>

<body onload="onloadFunction()">
<div id="target">
<p>target</p>
</div>
</body></html>
```

Вот какие функции выполняет этот код. В теле HTML определяется div со значением ID target. При загрузке документа функция onLoadFunction() применяет шаблон к данным при помощи метода parsetemplate(). Во время ее выполнения используются селекторы jQuery — сначала для того, чтобы найти шаблон, прикрепленный как скриптовый элемент с ID MyTemplate. Итогом синтаксического разбора шаблона будет расширенная строка (expanded string). Эта строка будет вставлена в div как внутренний HTML. Опять же для нахождения целевого div мы используем селектор jQuery.

Если сделать такой index.html в вашем предыдущем проекте и протестировать его, то на экране эмулятора будет следующая картина (рис. 17.13).

Итак, можно считать, что мы нашли инструмент, который эффективно поможет нам создавать сложные HTML-приложения. В этих простых примерах мы смогли показать далеко не весь функционал jQuery, но некоторые возможности мы все же продемонстрировали. В настоящее время в Интернете продолжается дополнение jQuery функциями, связанными с пользовательскими интерфейсами, — возможно, вас заинтересует этот процесс. Некоторые программисты, с которыми беседовали авторы, очень положительно высказывались об Aptana Studio, которая работает на базе Eclipse и предлагает множество новых разработок кода для JavaScript.

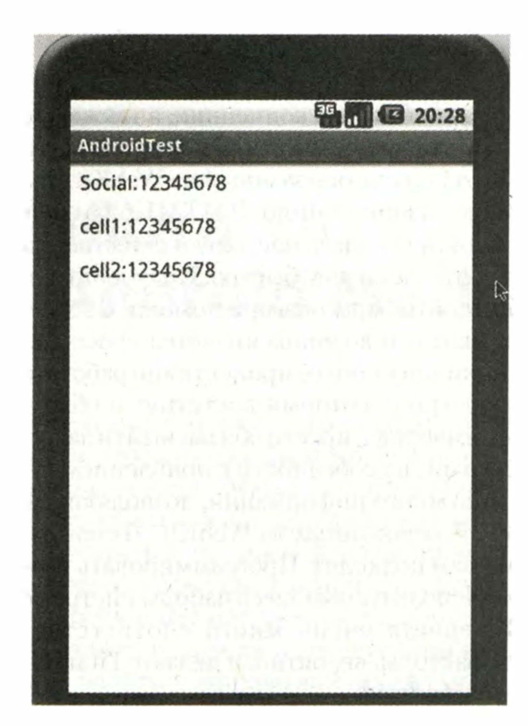


Рис. 17.13. Движок для создания микрошаблонов в Android

Завершая эту тему, кратко перечислим специфичные для Titanium обертки, применяемые при работе с JavaScript.

Дополнительные мобильные API для Titanium

В Titanium Mobile поддерживаются дополнительные API для работы с нативной платформой. Они перечислены по адресу http://www.codestrong.com/timobile/api/. Мы кратко опишем их в табл. 17.1.

Таблица 17.1. API Titanium для работы с Android

Пространство имен	Содержимое
API	Включает все методы сбора данных
Accelerometer	Может отслеживать события акселерометра и реагировать на них
App	Здесь можно получать свойства проекта во время исполнения
Database	Позволяет работать с базой данных SQLite
Filesystem	Может работать с локальными файлами и каталогами
Geolocation	Получает значения широты, долготы и следит за определенным местом
Gesture	Работает с видами в книжном и альбомном форматах
Media	Может работать с изображениями, звуком, видео
Network	Базовый сетевой стек, связанный с httpclient
Platform	Может работать, в частности, с телефонным номером, моделями, названиями, версиями и т. д.
UI	Обслуживает диалоговые окна, меню, таблицы

Резюме

В этой главе мы рассмотрели базовые концепции, а также представили инновационный метод, который существенно обогатит инструментарий программиста, работающего с Android. Этот подход, основанный на WebKit, позволяет несколькими способами расширить фреймворк Android. В HTML благодаря простой семантике шаблонов (например, прокрутка здесь настолько естественна, что вы ее просто не замечаете) разработка протекает очень быстро. Полученный в итоге пользовательский интерфейс можно легко оформлять при помощи CSS. Этот подход привлекателен еще и тем, что от начала и до конца является кросс-платформенным, таким образом, веб-разработчики могут более продуктивно работать на мобильных платформах. Наконец, тот прогресс, который достигнут в области пользовательских интерфейсов на веб-фреймворках, просто обязан выйти за их пределы и обогащать программирование и дальше, в особенности с появлением HTML 5.

В этой главе было дано много информации, позволяющей понять импликации и архитектуру технологий, основанных на WebKit. Теперь вы можете оценить, насколько эти технологии вам подходят. Программировать при помощи этой технологии очень интересно. Используемые здесь наборы инструментов очень популярны. К тому же в Интернете очень много соответствующей специальной документации. Все эти факторы, вероятно, и делают Titanium таким популярным при программировании в Android.

18 Работа с Android Market

Создавая отличную программу, которая должна всем понравиться, необходимо также позаботиться и о том, чтобы людям было несложно найти ее и скачать. Именно для этого Google и создал Android Market. Пользователь может просто нажать ярлык, находящийся в устройстве, и попасть прямо на этот рынок — а потом искать, просматривать и скачивать программы. Многие программы бесплатны, но не все; на рынке есть механизм оплаты, упрощающий приобретение программ. Рынок также доступен для намерений, работающих внутри приложения, и приложения могут выходить на этот рынок, руководя пользователем и помогая ему найти то, что нужно для успешной работы с конкретной программой. Например, если выходит новая версия вашей программы, пользователь может просто перейти на соответствующую страницу рынка и приобрести (или получить) новую версию. Разумеется, Android Market — не единственный путь, используя который программы попадают в устройства, в Интернете появляются и другие каналы.

Хотя выход на Android Market обычно доступен с любого коммерческого мобильного устройства, на рынок нельзя попасть через Интернет, а также из эмуляторов. Поэтому для разработчика ситуация несколько осложняется. В идеальном случае у вас должно быть собственное устройство с выходом на Android Market. Кроме того, на Android Market можно попасть с Android Developer Phone, однако в таком случае вы не сможете увидеть и приобрести ни одно платное приложение. Это один из применяемых в Google способов для борьбы с компьютерным пиратством.

В этой главе мы расскажем, как подготовиться к публикации приложения на рынке, как оформить его для продажи на рынке, как пользователи могут находить, скачивать и применять ваши программы, и, наконец, опишем альтернативные способы получения и приобретения программ Android, не связанные с Android Market.

Приступаем к публикации

Перед загрузкой программы на Android Market нужно получить статус публикатора. Для этого создается учетная запись разработчика (Developer Account). Имея такой аккаунт, вы сможете загружать свои программы на рынок, так чтобы их находили и скачивали пользователи. Google сделал процесс получения учетной записи разработчика относительно несложным и недорогим.

Чтобы опубликовать что-либо, нужно сначала создать аккаунт Google, например почтовый аккаунт на gmail.com. Затем нужно завести идентификатор на Android Market. Эта операция делается здесь: http://market.android.com/publish/signup. От вас потребуется указать имя разработчика, адрес электронной почты, адрес сайта и ваш контактный телефонный номер. Все эти поля необходимы для заполнения. Позже вы сможете изменить эти значения, если понадобится. Далее откроется экран, предлагающий заплатить регистрационный взнос. Эта операция осуществляется системой Google Checkout. Чтобы продолжить транзакцию, нужно будет войти под своим именем в Google.

Один из вариантов, предлагаемых в процессе платежа, — Keep my email address confidential (Сохранить почтовый адрес в секрете). Эта команда относится к актуальной транзакции, происходящей между вами и Google Android Market, где вы приобретаете право доступа для публикаций. Если выбрать Yes (Да), то ваш почтовый адрес будет сохранен в секрете от Google Android Market. При этом потенциальные покупатели будут видеть ваш адрес. Подробнее об этом — немного дальше.

После сбора платежных данных все почти готово. Далее следует дистрибьюторский договор, заключаемый между разработчиком и Android Market. Это юридический контракт между Google и вами. В нем описываются правила распространения приложений, сбора и возмещения платежей, обратной связи, оценивания, права пользователя, права разработчика и т. д. Подробнее об этом — в следующем разделе.

После заключения договора система обычно открывает страницу, называемую консолью разработчика (http://market.android.com/publish/Home).

Выполнение правил

В дистрибьюторском договоре указаны некоторые правила. Перед тем как заключить договор, можете проконсультироваться с юристом. В этом разделе мы опишем наиболее важные аспекты.

- Для работы с Android Market разработчик должен быть правильно зарегистрирован. Это означает, что для регистрации потребуется выполнить описанный выше процесс, принять соглашение и соблюдать его правила. При нарушении правил ваш аккаунт может быть заблокирован, а ваши товары удалены с рынка.
- Продукция может распространяться бесплатно или платно. В соглашении предусмотрены оба варианта. Чтобы продавать программы, вам понадобится платежная система, например Google Checkout. На момент выхода версии Android 2.0 Google Checkout была единственным инструментом для получения платежей на рынке. Эта система просто выставляет стоимость скачанных программ пользователю дополнительно к счету за телефон. T-Mobile анонсировала такую функцию 4 ноября 2009 года. На тот момент при работе с Android Market нельзя было пользоваться PayPal или другими платежными системами. Но, возможно, такие способы будут предусмотрены в следующих версиях. За платные программы будет взиматься операционный сбор, а также в некоторых случаях — взнос для курьера, выводимый из продажной цены. По состоянию на январь 2010 года операционный сбор составлял 30 %, то есть если вы продаете программу за $10, то вам достается $7, а Google — $3 (без учета сборов за доставку).

- Вы обязаны самостоятельно уплачивать налоги в соответствующие налоговые органы. Создавая счет продавца, вы указываете соответствующие налоговые ставки, которые будут учитываться при продаже программ клиентам в других регионах. Google Checkout будет собирать налоги в зависимости от заданных настроек. Эти деньги будут переводиться вам, а вы должны будете вносить их в качестве налога. Дополнительная информация об уплате налогов в США содержится на следующих сайтах: http://biztaxlaw.about.com/od/businesstaxes/f/onlinesalestax.htm и http://www.thestc.com.
- Вы сможете распространять бесплатную демоверсию программы с возможностью оплаты и последующего предоставления полного набора функций программы. Но платежи нужно будет собирать через официальную платежную систему Android Market. Такие комиссионные платежи — очень хороший способ борьбы с пиратством, кроме того, они позволяют оптимизировать общий наличный оборот. Однако если вы закачаете демоверсию, то продавать напрямую с Android Market полную версию вы не сможете. Возможно, такая функция появится в будущих версиях.
- Часто уплаченные деньги возвращаются пользователям, которые деинсталлируют программу в течение 48 часов после скачивания. Это своего рода бесплатное предоставление программы на пробу. Работа с бесплатным приложением, а также обновление платного могут быть проблематичными, и способ возврата денег через 48 часов — один из вариантов упрощения этого процесса (ниже в этой главе будут показаны и другие способы). Плата не возвращается пользователям, которые могли предварительно просмотреть программу перед приобретением. Это касается в том числе рингтонов и фоновых рисунков для рабочего стола. Программе требуется адекватная поддержка. При ее отсутствии пользователь может потребовать возврата уплаченных денег, возможно, вместе с платой за обслуживание.
- Пользователи могут сколько угодно раз устанавливать программу, скачанную с Android Market. Если пользователь возвращает устройство к заводским настройкам, эта функция позволяет возобновить работу всех приложений, не покупая их заново.
- Разработчики соглашаются защищать частную сферу и законные права пользователей. Это подразумевает защиту всех данных, которые могут быть собраны в процессе работы с приложением. Можно изменить правила, касающиеся защиты пользовательских данных, но только представив пользователю дополнительное соглашение между ним и вами, которое он должен принять.
- Ваша программа не должна конкурировать с Android Market. Google не нужны приложения, которые будут продавать продукцию для Android в обход рынка, а значит — и платежной системы. Это не запрещает вам продавать свои продукты и по другим каналам — просто сама ваша программа не должна быть предназначена для продажи продукции Android вне рынка.
- Google присваивает вашим продуктам собственные рейтинги. Рейтинг может зависеть от реакции пользователей, частоты установки конкретной программы, ее удаления, возврата денег и/или общего счета разработчика (Developer Composite Score). Google вычисляет Developer Composite Score на базе общей

истории программ данного разработчика, и это может влиять на рейтинги его новых продуктов. Поэтому важно, чтобы с вашим аккаунтом были связаны только хорошие программы, даже если они бесплатные. Выше мы описали, как предоставлять программу пользователю на пробу на 48 часов. Но так как возврат денег будет негативно сказываться на рейтингах ваших программ, это серьезная причина, по которой испытательный сок следует устраивать каким-то иным образом.

- При продаже программы на Android Market вы предоставляете пользователю «неэксклюзивную, действующую во всем мире, бессрочную лицензию на эксплуатацию и отображение продукта на устройстве». Однако вы можете написать и отдельное лицензионное соглашение с конечным пользователем (EULA), которое заменяет то, что указано выше. Соглашение EULA можно разместить на сайте или познакомить с ним покупателей другим способом.
- Google требует от вас придерживаться правил брендинга, принятых в Android. Они включают ограничения на использование названия Android, а также рисунка с роботом, логотипа и патентованного шрифта. Более подробно эти детали описаны по следующему адресу: http://www.android.com/branding.html.

Консоль разработчика

В консоли разработчика (Developer Console) можно купить специальное устройство для разработки программ — Android Developer Phone (ADP). На нем можно настроить аккаунт продавца (merchant account), закачивать программы и получать информацию об уже закачанных приложениях. Можно редактировать детали аккаунта, в том числе изменять имя разработчика, адрес электронной почты, адрес сайта и телефонный номер.

Android Developer Phone — это специальное устройство, созданное для разработчиков. Это полнофункциональный аппарат, который разблокирован и не связан ни с каким конкретным мобильным оператором. Он принимает любые SIM-карты, в состав устройства входят карта памяти размером 1 Гбайт, камера, скользящая клавиатура и GPS. «Разблокирован» в данном случае означает, что с устройством можно выполнять любые операции, в том числе загружать новые версии прошивки и платформы Android, а не только программы. Хотя может быть заманчиво приобрести такое устройство для тестирования программ, но лучше купить настоящий смартфон у коммерческого оператора. Программы, имеющиеся на ADP при поставке, — базовые, а в пакете от оператора содержится больше программ и функций. На коммерческий телефон, как и на ADP, можно выйти с рабочей станции, чтобы произвести отладку. Если вы хотите протестировать новые версии прошивки Android либо самой платформы Android, то вам нужен ADP. В других случаях вам больше подойдет коммерческий телефон Android.

Если в Google Checkout вы не создадите аккаунт продавца, то не сможете брать плату за свои программы на Android Market. Настроить аккаунт продавца совсем не сложно. Щелкните на ссылке, находящейся на консоли разработчика, заполните заявку, согласитесь с условиями использования — и все готово. Вам потребуется указать федеральный налоговый ID (ИНН), номер кредитной карточки, а также (в США) — номер социального страхового свидетельства (SSN). Налоговая ин-

формация используется для проверки вашего кредитного статуса, чтобы гарантировать своевременное внесение средств. Информация о кредитной карте применяется для обработки обратных платежей, если такая необходимость возникнет при спорах с покупателем, а на вашем аккаунте Google Checkout будет недостаточно средств. Можно также предоставить информацию о банковском счете, чтобы обеспечить электронный перевод средств в процессе продаж. Обратите внимание — служба Google Checkout работает не только на Android Market. Следовательно, не удивляйтесь, если Google Checkout предъявит вам информацию о транзакциях, проведенных за пределами Android Market. Выше упоминалось, что за обработку транзакций Android Market забирает 30 % прибыли. Если сделки совершаются за пределами Android Market, Google Checkout взимает отдельный сбор, не применяемый к продажам в рамках Android Market.

Основные функции консоли разработчика, которыми вы будете наиболее активно пользоваться, — загрузка программы и наблюдение за ней. (О загрузке программ поговорим ниже в этой главе.) Что касается наблюдения, Android Market представляет следующую информацию: общее количество скачиваний программы, а также сведения о том, у какого количества пользователей программа все еще установлена. Общий рейтинг программ оценивается в звездочках, от 0 до 5. Рейтинг зависит и от того, сколько пользователей оценили вашу программу. Консоль разработчика позволяет публиковать программу повторно, например после внесения обновлений, либо снимать ее с рынка (unpublish). При снятии с рынка программа не удаляется с частных устройств, а во многих случаях — и с серверов Google, особенно если это платное приложение. Если пользователь заплатил за вашу программу, а затем деинсталлировал ее и не потребовал вернуть деньги, он может переустановить ее позже, даже если вы убрали программу с рынка. Google закрывает пользователям всякий доступ к программе, только если вы нарушаете правила.

Комментарии к вашей программе вы можете просмотреть на Android Market так же, как и обычные пользователи. В ваших интересах читать эти комментарии и быстро устранять возникающие проблемы.

Подготовка приложения для продажи

Между появлением первых строк кода программы и выходом ее на Android Market проходит немало времени и много этапов. В этом разделе мы расскажем, что нужно сделать, готовя приложение для продажи.

Тестирование для различных устройств

На рынке появляется все больше новых устройств для работы с Android, и каждое из них, как правило, имеет новую конфигурацию оборудования. Очень важно тестировать свои программы на тех устройствах, для которых вы их пишете. Идеальный случай — найти устройство каждого из видов и протестировать продукт прямо на нем. Другой хороший вариант — конфигурировать AVD (Android Virtual Device) для каждого типа устройства, указать необходимые свойства оборудования, а потом при помощи эмулятора протестировать программу для каждого виртуального устройства. В Android SDK есть класс `Instrumentation`, облегчающий тестирование;

также рекомендуем инструмент UI/Application Exerciser Monkey. С их помощью можно настроить автоматическое тестирование, сэкономив немало времени. Перед началом тестирования вам, возможно, потребуется удалить все ненужные детали (артефакты), оставшиеся от предыдущих этапов тестирования, из кода и из каталога /res. Всегда нужно делать программу как можно более миниатюрной и быстрой, чтобы при этом она потребляла как можно меньше памяти.

Поддержка различных размеров экрана

Когда вышел Android SDK 1.6, разработчикам пришлось столкнуться с новыми размерами экранов, и чтобы программы работали на новых, маленьких дисплеях, стал применяться новый элемент <supports-screens> — дочерний по отношению к <manifest> в файле описания AndroidManifest.xml. Без этого нового тега, указывающего, что ваша программа работает и на маленьких экранах, она не будет отображаться на Android Market для устройств, имеющих такие экраны. Конечно же, мы подразумеваем, что программу нужно компилировать в Android SDK 1.6 или выше. Если вы хотите, чтобы программа работала и на тех устройствах, где все еще применяется Android SDK 1.5, то вам придется отказаться от всех преимуществ, обеспечиваемых новыми API, появившимися в Android SDK 1.6 или выше. В таком случае тестируйте программу в AVD, соответствующих как новым, так и старым устройствам. Для работы с разными размерами экранов в каталоге /res нужно создавать соответствующие ресурсы в расчете на каждый размер. Например, для файлов, находящихся в /res/layout, нобходимо создавать аналогичные файлы в /res/layout-small для работы с маленькими экранами. При этом не нужно создавать аналогичные файлы в /res/layout-large и /res/layout-normal, так как, если Android не найдет нужной информации в частной директории, например в /res/layout-large, он будет искать в более общей /res/layout. Вы можете также использовать комбинации классификаторов для файлов ресурсов. Например, /res/layout-small-land будет содержать шаблоны для небольших экранов альбомного формата. Для маленьких экранов, вероятно, придется создавать альтернативные экземпляры отрисовываемых объектов, а также специальные каталоги с ресурсами, учитывая как размер, так и разрешение экрана.

Подготовка AndroidManifest.xml для закачки

Возможно, файл AndroidManifest.xml придется немного изменить перед загрузкой программы на Android Market. По умолчанию ADT (инструменты для разработки в Android) в Eclipse не добавляет никаких атрибутов к тегу <application> в файле AndroidManifest.xml. Однако в этом теге перед загрузкой на рынок нужно обязательно указать атрибут android:icon. Обычно ADT помещает атрибут android:icon в тег <activity>, который нам также понадобится. На самом деле программа будет работать и в тех устройствах или эмуляторах, где атрибут android:icon находится в теге <activity>, но когда Android Market будет проверять вашу программу при закачке, он попытается найти информацию о пиктограмме в теге <application>. Поэтому просто скопируйте атрибут android:icon из тега <activity> в тег <application>. Кроме того, Android Market не допускает загрузки программ, имя пакета в которых

начинается с om.google, com.android, android или com.example. Надеемся, что вы не будете пользоваться такими пакетами в своих программах.

При тестировании программы на различных устройствах придется решить еще немало вопросов, связанных с совместимостью. В некоторых устройствах есть камеры, на других нет реальных клавиатур, есть только виртуальные, в некоторых используются не тачпады, а шаровые манипуляторы (трекболлы). В файле AndroidManifest.xml следует использовать теги <uses-configuration> и <uses-feature>, где указывается, какие требования к оборудованию и платформе предъявляет ваша программа. Android Market требует исполнения этих правил и не позволит скачать вашу программу на устройство, которое не может обеспечить необходимой поддержки. Обратите внимание: эти теги отличаются от тегов <uses-permission> того же файла AndroidManifest.xml. Ведь если в устройстве есть камера, это еще не означает, что пользователь хочет предоставить доступ к ней вашей программе. В то же время требование, согласно которому вашей программе необходим доступ к камере, еще не означает для Android Market, что в устройстве, которое скачивает программу, обязательно должна быть камера. В большинстве случаев в файле AndroidManifest.xml понадобится указать оба тега — свидетельствующих, что требуется как камера, так и доступ к ней.

Локализация приложения

Если ваша программа будет использоваться в разных странах, можно подумать о ее локализации. Технически это делается относительно легко. Другое дело — найти специалиста для локализации. С технической точки зрения при локализации просто создается еще один подкаталог в /res. Например, в /res/values-fr будет содержаться французская версия файла strings.xml. Для перевода строковых значений возьмите имеющийся файл strings.xml, а затем сохраните новый (переведенный) файл в новом каталоге ресурсов под тем же именем, что и оригинал. Этот же метод работает и с другими типами ресурсных файлов, например с отрисовываемыми объектами и меню. Изображения и цвета могут быть более привлекательными для пользователей, если учитывать культурную специфику того или иного цвета или символа. Поэтому, указывая в качестве ресурсов цвета, избегайте пользоваться названиями цветов. В онлайн-документации по цветовому оформлению часто можно встретить подобные записи:

```
<color name="solid_red">#f00</color>
```

Это значит, что в коде или других файлах ресурсов вы ссылаетесь на название конкретного цвета, в данном случае — solid_red. Чтобы выбрать другой цвет, более приемлемый в иной стране или культуре, лучше называть цвета как accent_color1 или alert_color. В английском контексте для этой цели подходит красный, но, например, в Испании лучше применить определенный оттенок желтого. Благодаря записи alert_color вы не указываете прямо, каким цветом пользуетесь, поэтому при фактическом изменении одного цвета на другой будет меньше путаницы. В то же время можно разработать красивую гамму из строго определенных цветов и оттенков, и в таком случае лучше будет использовать в нужных местах именно определенные цвета.

В разных странах можно изменять названия или количество элементов меню либо по-разному строить меню в зависимости от того, где именно применяется программа. Если вы окажетесь в такой ситуации, то, вероятно, лучше будет поместить текст всех строк в `strings.xml` либо в другие файлы, расположенные в каталоге `/res/values`, и во всех других файлах ресурсов использовать подходящие строковые ID. Таким образом значительно снижается вероятность попадания строкового значения в какой-то неподходящий файл ресурсов. Языковые файлы будут применяться только к тем файлам, которые находятся в `/res/values`.

Подготовка ярлыка для вашего приложения

После того как покупатели и пользователи скачают программу, они будут видеть ее название и ярлык, гордо красующиеся и на Android Market, и на их устройстве. Поэтому рекомендуем очень тщательно подходить к выбору ярлыка и названия. Локализуйте их, если нужно. И помните, что экраны устройств бывают разного размера, поэтому для некоторых устройств ярлык, возможно, придется подкорректировать. Посмотрите, какие ярлыки у программ других авторов, особенно в той же категории, что и ваша. Вы же хотите, чтобы ваша программа была заметной, поэтому не дайте ей затеряться среди аналогов. В то же время необходимо, чтобы название и ярлык удобно располагались на главном экране устройства, среди ярлыков программ, выполняющих другие функции. Не стоит делать ярлык совсем не подсказывающим, какие функции выполняет программа, иначе пользователь может запутаться.

Размышления относительно платных приложений

Если программа является бесплатной, никто даже не будет пытаться распространять ее пиратские копии, но вот с платными приложениями ситуация иная. Если вы продаете вашу программу, то вам придется подумать о решении связанных с этим вопросов. Захотите ли вы отделить друг от друга платные и бесплатные приложения? В таком случае придется писать и поддерживать два приложения. Либо вы оставите базовый код одинаковым и при помощи того или иного метода будете узнавать, оплачена ли данная копия программы. Независимо от того, какой путь вы выберете, как вы защитите программу от незаконного копирования и инсталляции на других устройствах? Поскольку телефоны полны уязвимостей, а злоумышленники умеют взламывать устройства, обеспечить защиту программы от копирования очень сложно. Один из способов поддержания единой базы кода с одновременным разделением программ на платные и бесплатные — пользоваться `PackageManager`:

```
this.getPackageManager().checkSignatures(mainAppPkg, keyPkg)
```

Этот метод сравнивает цифровые подписи двух именованных пакетов и возвращает `PackageManager.SIGNATURE_MATCH`, если обе подписи существуют и являются одинаковыми. У всех программ, одновременно находящихся на рынке, имена пакетов должны быть разными, и это хорошо. В коде (если нам приходится решать, допускать ли использование определенной функции или нет) мы вызываем этот метод, который, в свою очередь, предоставляет имя пакета нашего основного при-

ложения, а также имя пакета разблокировочного приложения. После этого программа-разблокировщик на рынке делается платной. Если пользователь купит разблокировочную программу и установит ее в устройстве, основная программа получит совпадающую подпись и откроет дополнительные функции. Менее аккуратный способ работы с единой кодовой базой — применять системы контроля версий исходного кода и конфигурировать необходимое совместное использование общих элементов, а также писать скрипты для создания бесплатных и коммерческих версий приложения.

Возвращение пользователей обратно на рынок

В Android недавно была введена новая схема URI, помогающая находить программы на рынке Android Market: `market://`. Например, если вы хотите направить пользователя на рынок, где он может найти недостающий программный компонент, либо предложить ему купить дополнительную программу-разблокировщик, то нужно написать код, похожий на приведенный ниже, где `MY_PACKAGE_NAME` следует заменить реальным именем пакета:

```
Intent intent = new Intent(Intent.ACTION_VIEW,
      Uri.parse("market://search?q=pname:MY_PACKAGE_NAME"));
startActivity(intent);
```

Так в устройстве запускается программа Market, которая переадресует пользователя к пакету с таким именем. Затем пользователь может решить, покупать ли программу либо просто скачать. Обратите внимание — эта схема не работает в обычном браузере. Кроме поиска по имени пакета (`pname`), можно производить поиск по имени разработчика при помощи схемы `market://search?q=pub:"FnameLname"`, а также по тексту из любого открытого поля (по названию приложения, имени разработчика, описанию приложения) при помощи схемы `market://search?q=<querystring>`.

Подготовка APK-файла для закачки

Чтобы подготовить готовое приложение к выставлению на рынок, то есть создать файл APK для закачки, нужно выполнить следующие операции (все они рассмотрены в главе 7, в подразделе «Подписывание приложений для развертывания» раздела «Модель обеспечения безопасности в Android»).

1. Создать сертификат продукта, при помощи которого вы будете подписывать программу (если еще не сделали этого).
2. Если работаете с картами — заменить ключ MAP API в файле `AndroidManifest.xml` на MAP API вашего продукта. Если вы забудете это сделать, пользователи не смогут даже увидеть карты.
3. Экспортировать программу, щелкнув правой кнопкой мыши на проекте в Eclipse и выбрав **Android Tools ▸ Export Unsigned Application Package** (Инструменты Android ▸ Экспортировать неподписанный пакет приложения), а затем выбрать нужное имя файла. Удобно давать этому файлу временное имя, так как после запуска `zipalign` на этапе 5 нужно будет указать имя итогового файла, а это должно быть название файла APK готовой программы.

4. Применить jarsigner к новому файлу APK, чтобы подписать его сертификатом, упомянутым в шаге 1.
5. Применить zipalign к новому файлу, чтобы уложить все незаархивированные данные в имеющемся объеме (границах) памяти и обеспечить лучшую работу программы во время исполнения. На этом этапе вы присваиваете окончательное название файлу APK вашей программы.

Закачка вашего приложения

Закачивать программы на рынок просто, но для этого требуется некоторая подготовка. Прежде чем начать закачку, нужно подготовить несколько элементов и решить пару вопросов. В этом разделе мы рассмотрим необходимые решения и приготовления. Затем, когда у нас будет все, что нужно, мы перейдем в консоль разработчика и выберем вариант Upload Application (Закачать приложение). Система подскажет вам, какой набор данных о программе нужно указать, а рынок совершит ряд операций с вашей программой и этой информацией — и вот после этого программа появится на рынке!

В предыдущем разделе мы поговорили о том, что нужно сделать, чтобы подготовить к закачке файл APK. Но чтобы программа стала привлекательной для потенциальных покупателей, вам придется стать маркетологом. Нужно хорошо описать, что это за программа и что она делает, а также представить удачные изображения, чтобы покупатель представлял, что именно он скачивает.

В начале процесса закачки приложения от вас обязательно потребуются его скриншоты. Проще всего сделать скриншоты программы при помощи DDMS. Запустите Eclipse, а затем переключите эмулятор в режим DDMS и вид Device (Устройство). Из вида Device (Устройство) выберите устройство, в котором работает ваша программа, а затем нажмите кнопку Screen Capture (Снимок экрана). Она похожа на маленькую картинку и расположена в верхнем правом углу. Кроме того, команда Screen Capture (Снимок экрана) есть в меню View (Вид). При сохранении изображения выберите 24-битный цвет. Android Market преобразует скриншот в сжатый файл JPEG. Если сжимать 24-битные цвета, результат получится лучше, чем при сжатии 8-битных. Выберите скриншоты, которые позволят вашей программе выделиться на фоне остальных, но помните, что на этих скриншотах следует показать наиболее важные функции.

Можно также представить проморисунок, но по размеру он будет меньше, чем скриншот. Хотя такой рисунок не является обязательным, все же рекомендуем его дать. Нельзя угадать, когда рисунок удастся отобразить, а когда — нет. Но если рисунка не будет, то вы не будете знать, что пользователь увидит вместо него.

Android Market потребует от вас текстовое описание программы, которое будут видеть пользователи. В том числе к текстовой информации относится заголовок, текст описания и промотекст, который можно указать, только если вместе с ним вы приводите проморисунок. Текст можно дать на нескольких языках, так как, вероятно, вы собираетесь распространять программу сразу во многих странах. Рисунки, описанные выше, можно загрузить на Android Market только один раз, поэтому, если ваши скриншоты должны по-разному выглядеть в различных регио-

нах, нужно придумать другие способы донести их до покупателя, возможно — разместить на своем сайте. В перспективе эта ситуация может измениться.

Если вы написали для пользователей отдельное лицензионное соглашение, в тексте аннотации к программе разместите ссылку на него, чтобы пользователь мог сначала прочитать это соглашение и решить, загружать ему программу или нет. Учтите, что пользователи, пытающиеся найти программу, скорее всего, будут применять поисковую функцию, поэтому правильно подберите слова для аннотации, чтобы как можно больше поисковых запросов относилось к функциям вашей программы. Наконец, к тексту стоит присовокупить краткий комментарий с адресом вашей электронной почты и предложить пользователю обратиться к вам, если возникнут проблемы. Не получив такой простой подсказки, клиенты часто оставляют отрицательные комментарии, а такие комментарии достаточно усложняют процесс нахождения неисправностей и их устранения. В то время как электронная переписка с пользователем вам только поможет.

У механизма обратной связи с пользователем, описанного выше, есть один недостаток — в нем не различаются версии программы. Если негативный отзыв относится к версии 1 и в версии 2 вы исправите обнаруженные недостатки, комментарии к версии 1 никуда не денутся и пользователь не поймет, что такой комментарий касается старой версии. Кроме того, при выпуске новой версии программы ее рейтинг (количество звезд) не сбрасывается. Не забывайте об этом, создавая свой маркетинговый текст, либо подумайте о выпуске новой версии в виде отдельной программы.

Одна из ваших задач при написании такого текста — объяснить, какие права доступа потребуются для работы с приложением. Имеются в виду права доступа, которые вы указываете в тегах `<uses-permission>` файла `AndroidManifest.xml`. Когда пользователь ставит программу на устройство, Android проверяет файл `AndroidManifest.xml` и просит пользователя предоставить все права доступа, а лишь потом завершает установку. Поэтому и вы можете рассказать об этих правах заранее. В противном случае вы рискуете получить негативные отзывы от пользователей, которые будут неприятно удивлены, если программа потребует от них право доступа, которое они не готовы предоставить. И это уже не говоря о возврате денег, который отрицательно отразится на вашем общем счету разработчика. Наряду с правами доступа ваша программа может требовать определенного типа экрана, наличия камеры либо другой характеристики устройства — обо всем этом нужно написать в аннотации к приложению.

При закачке необходимо выбрать для программы тип и категорию. По умолчанию все программы бесплатны, и если вы собираетесь продавать ваш продукт, то предварительно нужно создать в платежной системе Google Checkout аккаунт продавца. Установить для программы разумную цену нелегко, если только вы не обладаете талантом к рыночным исследованиям (правда, и в таком случае придется потрудиться). Если цена слишком высока, она может отпугнуть покупателя, а если покупатель попробует поработать с программой и решит, что она не стоит запрошенной цены, вы будете вынуждены вернуть деньги. Слишком низкие цены также могут сослужить медвежью услугу — покупатель может решить, что такая дешевая программа вряд ли действительно хороша.

В Android Market предусмотрена функция защиты от копирования программы при загрузке. В таком случае программа будет потреблять больше памяти. Правда,

она не защищена от случайных ошибок, поэтому нет гарантий, что вашу программу все же не скопируют с устройства. Поэтому для борьбы с пиратством можно придумать дополнительные или альтернативные способы.

Одно из последних решений, которое следует принять перед загрузкой, — выбрать регионы и мобильных операторов, для которых ваше приложение будет видимым. Выбирая опцию All (Все), вы предоставляете программу всем. Однако можно ограничить доступ географически либо оставить его лишь для некоторых операторов. В зависимости от того, какие функции выполняет программа, могут потребоваться дополнительные ограничения, чтобы не нарушать экспортного законодательства. Вы можете применить ограничение по оператору, если у вашего приложения возникают проблемы совместимости с устройствами определенных операторов либо с их политиками. Чтобы просмотреть список операторов, щелкните на ссылке с названием страны — и увидите операторов, действующих в этой стране. Из их числа сможете выбрать тех, которые вас интересуют. При выборе варианта All (Все) Google автоматически отображает вашу программу для всех новых операторов, появляющихся на рынке, а также во всех новых регионах — вашего вмешательства не требуется.

Хотя в профиле разработчика и содержится ваша контактная информация, при загрузке каждой новой программы вы также можете указывать новые данные. На рынке в качестве контактной информации, относящейся к конкретной программе, запрашивается адрес сайта, адрес электронной почты и телефонный номер. Нужно указать как минимум один контакт из этих трех, чтобы пользователи могли запросить поддержку, но все три контакта можно не указывать.

Когда все эти решения будут приняты, вы должны, во-первых, засвидетельствовать, что ваша программа выполняет предъявляемые Android требования к контенту (не допускается никакой эротической информации), а во-вторых, указать, что программа может продаваться как экспортный товар. Не забывайте: всегда есть возможность распространять программу и по другим каналам, а не только через Android Market. Затем можно опубликовать приложение, нажав кнопку Publish (Опубликовать). Android Market проверит ваше приложение по нескольким параметрам, в частности посмотрит дату истечения сертификата. Если все будет нормально, то с этого момента пользователи смогут ее скачать. Поздравляем!

Работа пользователя с Android Market

Официально Android Market доступен только с мобильных устройств. Это означает, что пользователь будет работать с рынком только через такое устройство. Разработчики никак не контролируют работу Android Market, они могут только сопроводить свои программы, появляющиеся на рынке, хорошими аннотациями и красивыми картинками. Поэтому работа с программой с точки зрения пользователя во многом зависит от Google. С помощью устройства пользователь может искать программы по ключевым словам, посмотреть топ-листы недавно скачанных программ (как бесплатных, так и платных), изучить самые популярные, либо самые новые приложения или включить обзор по категориям. Когда пользователь находит интересующую его программу, он просто выделяет ее и перед ним всплывает окно

с детальной характеристикой приложения. В этом окне можно скачать или купить программу. При покупке пользователь переходит в платежную систему Google Checkout, где выполняются финансовые операции. После скачивания новая программа отображается в списке вместе со всеми остальными.

На Android Market предусмотрена функция просмотра загруженной программы в разделе My Downloads (Мои загрузки). В этом окне перечисляются все установленные приложения, а также все приобретенные вами программы, даже если некоторые из них вы уже удалили (возможно, удаляя их, вы просто освобождали место для новых программ). Это означает, что купленную программу вы можете удалить со смартфона, а затем повторно установить ее, не покупая заново. Разумеется, если вы попросите вернуть деньги, приложение не будет отображаться в My Downloads (Мои загрузки). Кроме того, здесь не будут показаны бесплатные программы, удаляемые из устройства. Список программ связан с вашим аккаунтом Google, используемым в устройстве. Это означает, что вы можете взять другое устройство и у вас по-прежнему сохранится доступ к тем программам, за которые вы заплатили. Но будьте внимательны. Поскольку в Google у вас может быть несколько профилей, то, чтобы в устройстве открылся доступ к программе, вы должны зайти на него именно с того аккаунта, под которым приобретали программу. При просмотре программ в My Downloads (Мои загрузки) около всех приложений, для которых доступны обновления, будет стоять соответствующий значок, и вы сможете скачать обновления.

Android Market производит фильтрацию программ, доступных для пользователей. Это делается несколькими способами. В некоторых странах пользователи могут видеть только бесплатные приложения, так как этого требуют местные законы. Google пытается обойти все препятствия, чтобы коммерческие программы были доступны по всему миру. Но пока такое время еще не наступило, в некоторых странах пользователи не будут иметь доступа к платным приложениям. Пользователи, работающие с устройствами, на которых стоят старые версии Android, не увидят программ, написанных при помощи более новых версий Android SDK. Если конфигурация устройства несовместима с характеристиками той или иной программы (это указывается в файле AndroidManifest.xml), пользователь не увидит такие программы. Например, если программа не предназначена для работы с очень маленькими экранами, то ее название не будет отображаться на устройствах, экран которых слишком мал. Как правило, такая фильтрация предназначена для того, чтобы пользователь не мог скачать программу, которая не будет работать на его устройстве.

Если покупать на Android Market программы из любых стран, кроме США, транзакция связана с конверсией валют, за что может взиматься дополнительный сбор. На самом деле вы приобретаете программу из страны продавца посредством Google Checkout. На Android Market отображается приблизительная сумма, и точный размер сборов может различаться, в зависимости от того, где происходит транзакция и через какую платежную систему. Иногда покупатель может заметить у себя в аккаунте отложенную транзакцию на небольшую сумму (например, $1). Google производит такие операции, чтобы гарантировать, что платежные реквизиты указаны верно, и на самом деле такая сумма не взимается.

В принципе, попасть на Android Market можно не только с мобильного устройства. Есть несколько сайтов, которые зеркалят Android Market. Покупатели могут искать на этих сайтах товар, просматривать категории и находить в Интернете

информацию о программах Android, даже не имея нужного мобильного устройства. На таких сайтах не действует фильтрация, которую Android Market включает в зависимости от вашего географического положения и характеристик устройства. Однако и скачать программу на устройство с такого сайта тоже нельзя. На Android Market не предусмотрено скачивание программ через веб, поэтому, даже если вы узнаете на таком сайте, что на рынке есть та или иная программа, вы все равно не сможете получить ее в обход рынка, если она не отображается у вас на устройстве. Примерами подобных зеркал являются сайты http://www.androlib.com и http://www.androidzoom.com. Был еще один такой сайт, http://www.cyrket.com, но на момент написания книги он не работал.

Кроме того, программы Android можно покупать в магазинах, никак не связанных с Android Market. Такими сайтами, например, являются http://www.andappstore.com, http://slideme.org и http://www.androidgear.com. На них предусмотрен поиск программ, просмотр категорий и изучение информации о приложениях, а также присутствует возможность скачивать программы как с устройства, так и через веб-браузер. Эти сайты не обязаны подчиняться правилам Google, в том числе что касается комиссионных сборов за платные программы и способов платежа. На этих сайтах для приобретения программ можно пользоваться PayPal и другими платежными системами. Кроме того, здесь не действуют ограничения географического характера и ограничения, связанные с конфигурацией устройства. На некоторых сайтах предлагается установить клиент для Android, в других случаях такие клиенты уже установлены в устройстве. Пользователь может просто включить браузер и через сайт найти и загрузить нужные ему программы. Когда файл уже сохранен в устройстве, Android знает, что с ним делать. Другими словами, Android воспринимает загруженный APK-файл как программу. Если открыть в браузере историю загрузок (Download History, не путайте с My Downloads), то сможете выбрать, что устанавливать, а что нет. Это значит, что вы можете настроить собственные методы для установки программ пользователями, в том числе разрешить установку программ с вашего сайта с применением удобного вам метода платежа. Так или иначе, вам придется собирать все налоги с продаж и передавать их соответствующим органам.

Хотя Google и не запрещает пользоваться такими методами, они не обеспечивают покупателю такой защиты, какую предоставляет Android Market. Возможно, приложение, купленное вне Android Market, не будет работать в устройстве покупателя. Кроме того, покупателю придется создавать резервные копии на случай, если программа будет удалена с устройства, либо для переноса программ на новые устройства. Не забывайте, что Google позволяет разработчикам продавать программы на нескольких рынках одновременно. Поэтому рассмотрите все варианты и выберите из них наиболее подходящий.

Резюме

Итак, вы готовы идти в большой мир со своими программами, написанными для Android! Мы рассказали, как к этому подготовиться, как приготовить свою программу, как ее опубликовать и как клиенты смогут найти, скачать и использовать ваше творение.

19 Обзор и ресурсы

В заключительной главе нашей книги мы рассмотрим, на какой стадии развития сейчас находится Android, и ее перспективы на рынке мобильных систем.

Чтобы оценить успех Android, достигнутый до сих пор, мы сначала перечислим компании, которые уже делают устройства для работы с Android. Возможности Android мы кратко опишем на примерах спецификаций T-Mobile G1 (с 2008 года), Motorola Droid (с конца 2009 года) и недавно появившегося разблокированного устройства Google NexusOne (с начала 2010 года). Кроме того, в течение 2009 года появилось немало интернет-магазинов, продающих программы для Android. Некоторые из этих магазинов мы также перечислим.

Чтобы представить, какое будущее ожидает Android, мы рассмотрим некоторые мобильные операционные системы и сравним их с Android и ее фреймворком. В конце главы мы перечислим полезные ресурсы, которые посвящены разработке для Android и новостям, связанным с ей.

Актуальное состояние Android

Android успела добиться немалых успехов. В конце 2008 года на рынке было всего одно устройство, работающее с Android, — T-Mobile G1. В начале 2009 года уже предполагалось, что к концу года производителей будет не менее 18. Это звучало довольно амбициозно. Еще в начале 2009 года насчитывалось всего несколько тысяч программ для Android. Когда прошел 2009 год, количество производителей действительно превысило 18, на рынке уже есть самые различные устройства, работающие с Android, — от сотовых телефонов до нетбуков и е-ридеров. Уже существует более 20 000 программ, продающихся в самых разных магазинах. В Wall Street Journal практически каждую неделю пишут статью об Android, а то и не одну.

Рассмотрим, какие устройства для работы с Android есть на рынке и какие вскоре должны появиться.

Компании, предлагающие мобильные устройства с ОС Android

По состоянию на конец 2009 года устройства, работающие под управлением Android, выпускались в том числе следующими компаниями:

- Archos (интернет-планшет);
- Barnes and Noble (электронная книга Nook Book);

- Entourage (двухсторонний е-ридер, очень похожий на настоящую книгу);
- General Mobile;
- HTC (Maker of Magic, Hero, Droid Eris, Click/Tatoo);
- HKC (клон HTC);
- Huawei;
- Lenovo;
- LG Group;
- Motorola;
- Qigi;
- Samsung;
- Gini;
- Ericsson;
- Acer;
- Skytone (нетбук альфа-680);
- ICD Vega (планшет).

Большинство этих компаний производит мобильные телефоны, некоторые — нетбуки (Acer) и электронные книги (Barnes and Noble, Entourage). Как видите, недостатка в устройствах для Android нет.

Рассмотрим некоторые из этих устройств, чтобы сориентироваться в их технических характеристиках. Начнем с Motorola Droid.

Motorola Droid

Motorola Droid имеет процессор ARM Cortex с частотой 256–550 МГц (согласно спецификации на сайте Motorolla). Оперативная память устройства — 256 или 512 Мбайт. Аппарат оборудован емкостным сенсорным экраном WVGA, а также жидкокристаллическим дисплеем TFT (тонкопленочный транзистор). На Droid установлена камера с разрешением 5 мегапикселов (в более специализированных цифровых камерах — до 12 мегапикселов). Droid поддерживает GPS, Wi-Fi, Bluetooth. Кроме того, в устройстве есть USB-вход, совместимый с USB 2.0. В Droid также поддерживается акселерометр, датчики близости, устройство распознает внешнее освещение. На Droid есть и клавиатура. Более подробная спецификация приведена по адресу http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/Mobile-Phones/ci.Motorola-DROID-US-EN.alt.

T-Mobile G1

Сравним Motorola Droid с T-Mobile G1, вышедшим в 2008 году. На G1 стоит процессор Qualcomm с частотой 528 МГц. Мощность оперативной памяти — 192 Мбайт. Устройство оборудовано тонкопленочным сенсорным экраном HVGA (320 × 480). Есть камера с разрешением 3,2 мегапиксела. Устройство может работать с Bluetooth, GPS и Wi-Fi. Кроме того, на нем есть микровход для USB. Более подробная спецификация — на сайте http://www.htc.com/www/product/g1/specification.html.

Nexus One

Приведем пример еще одного, более нового устройства от Google. В начале 2010 года Google выпустил разблокированный (работающий без SIM-карты) сотовый телефон Nexus One (http://www.google.com/phone) со следующими характеристиками. В нем используется процессор Snapdragon (модель Qualcomm QSD 8250) с частотой 1 ГГц (сравните с действующим процессором iPhone — 600 МГц и Droid — 550 МГц). В устройстве установлен дисплей WVGA с разрешением 800 × 480, вместо тонкопленочного транзистора (TFT) используется технология AMOLED (активная матрица на органических светодиодах), альтернатива TFT. Дисплеи на органических светодиодах дают более яркое, резкое и светлое изображение. Кроме того, на Nexus One есть 5-мегапиксельная камера. Клавиатура имеется только виртуальная. Устройство поддерживает GSM и Wi-Fi. Стоимость без контрактных расходов составляет $529. При подключении к определенному оператору, например T-Mobile, цена может быть иной. На момент написания книги с устройством Android можно было подключиться только к T-Mobile, но к весне 2010 года Verzion в США и Vodafone в Европе обещали также предоставить такую возможность. Более подробная техническая спецификация приведена по адресу http://www.google.com/phone/static/en_US-nexusone_tech_specs.html.

Как видите, Android неплохо развивается. Теперь взглянем на нее с другой стороны — с точки зрения покупки программ.

Магазины для покупки программ Android

Растет не только количество производителей оборудования, но и производителей программ для Android, а также число магазинов, продающих их онлайн. На момент написания книги в этот список входили:

- Android Market (от Google);
- Slideme;
- Andappstore;
- Mplayit;
- Androlib;
- Storeoid (от General Mobile);
- Androidgear;
- Handango.

Может возникнуть вопрос: а зачем так много магазинов? Причин немало, перечислим некоторые из них. По состоянию на сегодняшний день рынок Google доступен не во всех странах, кроме того, он поддерживает не все способы платежа. С другой стороны, некоторые устройства могут иметь специфические особенности, например работать с двумя SIM-картами (таково, например, DSTL1 от General Mobile). Определенные программы могут быть приспособлены для работы с такими устройствами. Еще одна причина — некоторые производители (Motorolla) или операторы (Verzion) желают сохранять контроль над своими программами.

Есть и еще одно критическое замечание: говорят, что на рынке Google не лучшим образом реализована функция обзора (browsing). Независимо от причин магазины плодятся как кролики.

Кратко охарактеризуем перечисленные выше магазины и их ассортимент.

Android Market (http://www.android.com/market/) от Google — это официальный магазин Android, который с каждой новой версией становится все лучше. В главе 18, посвященной Android Market, мы рассмотрели его сильные и слабые стороны.

Магазин **Slideme** (http://slideme.org/applications) основан в 2008 году, физически расположен в Сиэтле. Целью Slideme являются нишевые рынки. Он предлагает различные системы платежа, а также программы, которые сложно найти по другим каналам. Еще одна заявленная цель Slideme — глобальная продажа и доставка программ.

Целью **Andappstore** (http://andappstore.com/), по-видимому, является поддержка как официальных, так и неофициальных устройств с Android. Сайт достаточно безыскусен — по сравнению с ним Slideme кажется гораздо более проработанным.

На сайте **Mplayit** (http://mplayit.com) можно найти программы не только для Android, но и для iPhone и Blackberry. Mplayit — это приложение для Facebook, здесь очень удобно искать и покупать программы. Сайт хорошо сделан, на нем можно создавать пользовательские сообщества и делиться в них опытом. Этот сайт является своего рода каталогом, описывающим основные категории Android Market.

AndroLib (http://www.androlib.com/), как и Mplayit, — каталог, построенный по категориям, представленным на Android Market. Но в отличие от Mplayit он никак не связан с Facebook.

Storeoid создан General Mobile (http://www.generalmobile.com/), компанией, изготовляющей двухсимочные телефоны Android. В этом магазине General Mobile продает программы, специально предназначенные для ее сотовых телефонов.

Androidgear (http://www.androidgear.com) поддерживается PocketGear, серьезной компанией, организовавшей в веб внушительную платформу для продаж. Androidgear открыт специально с целью продажи программ для Android. Сайт хорошо организован.

Handango (http://www.handango.com) представляется надежной торговой точкой, предлагающей любые типы гаджетов и мобильных приложений. Здесь продаются программы не только для Android, но и для многих других мобильных систем.

Общее количество программ для Android, продававшихся во всех этих магазинах, к концу 2009 года составило около 20 000. Как видите, рынок Android быстро приближается к критической массе. Для сравнения — iPhone насчитывает около 80 000 программ.

Обзор Android

Android более чем оправдала ожидания, которые с ним связывались на протяжении 2009 года. В этом разделе будут рассмотрены конкуренты Android — другие имеющиеся на рынке мобильные операционные системы. Кроме того, мы затронем еще одну особенность Android, говорящую о ее быстрой адаптации к изменениям, — поддержку такого активно изменяющегося стандарта, как HTML 5.

Этот анализ позволит понять, что же такого особенного в Android, что позволило ей добиться успеха, и сохранится ли этот успех в будущем.

Быстрый обзор операционных систем, используемых в мобильных устройствах

Уже больше десяти лет идут разговоры о том, что мобильное программирование станет темой года. Но реальность показала, что технологии, связанные с мобильными устройствами, развиваются поступательно и не так быстро. До пришествия на рынок iPhone нельзя было сказать, что в аппаратном и программном обеспечении мобильных устройств происходили какие-то революционные изменения. В 2009 и 2010 годах продолжала расти вычислительная мощность оборудования и постепенно улучшалось качество изображения. В 2010 году уже появились устройства со скоростью процессора до 1 ГГц и объемом памяти от 1 до 4 Гбайт.

Поскольку мы рассматриваем ОС Android, возникает очевидный вопрос: какие еще операционные системы для мобильных телефонов существуют и чем они отличаются от Android. Разработка мобильных ОС продолжается нарастающими темпами. Из ОС можно назвать следующие: Symbian, Blackberry (RIM — Research In Motion), iPhone OS (Apple), Moblin (Intel), Maemo (Nokia), Windows Mobile (Microsoft), Palm OS BREW (Qualcomm) и JavaFx Mobile/SavaJe (операционная система на базе Java от компании Sun). Рассмотрим основные характеристики этих систем.

Blackberry OS (http://na.blackberry.com/eng/developers/) от компании RIM (Research in Motion) — очень популярная операционная система, поскольку устройства с Blackberry широко используются в корпоративном секторе. Эта операционная система является специализированной (dedicated). Среди поддерживаемых ОС Blackberry программируемых интерфейсов — Java ME и соответственно язык Java. Следует также отметить, что в устройстве поддерживается MIDP (профиль для мобильного устройства с информационными функциями) и WAP (проткол для беспроводного доступа) — в основном для поддежки доступа к Интернету с мобильного устройства.

Symbian (http://www.symbian.org/) — одна из ранних операционных систем, созданная для процессоров ARM и специализированная для мобильного рынка. Недавно Symbian приобретена компанией Nokia и используется в младших моделях их телефонов. После приобретения Nokia открыла доступ к коду Symbian (система стала open source). Основные языки, применяемые в Symbian, — C++ и Java (через Java ME).

Moblin (http://moblin.org/about-moblin) основана на Linux и поддерживается компанией Intel. Эта оптимизированная платформа Linux является свободно распространяемой, предназначена для работы с нетбуками и мобильными устройствами для выхода в Интернет. Среда разработки представляет собой набор инструментов Linux. Intel предоставляет набор библиотек, основанных на C, называемый Moblin Core и оптимизированный для мобильных платформ. К этим библиотекам можно обращаться на нескольких высокоуровневых языках. Пользовательский интерфейс Moblin основан на свободно распространяемом продукте, называемом clutter

(http://clutter-project.org), который, по существу, является оберткой (wrapper) OpenGL. Очень важный момент, связанный с Moblin, — Intel старается объединить все работы, ведущиеся в Linux и связанные с мобильными платформами, в одном флаконе.

Maemo (http://maemo.org/development/), как и Moblin, основана на Linux. В Maemo поддерживается разработка на языках C, C++ и Python с применением плагинов Eclipse. Maemo использует кросс-компиляционный инструмент scratchbox (http://www.scratch-box.org), позволяющий разрабатывать программы, которые могут работать на других процессорах, например ARM. Платформа Maemo была создана Nokia для своих высокотехнологичных устройств. Кроме того, здесь можно использовать GTK+, популярный инструментарий для написания оконных приложений в Linux.

iPhone OS (http://developer.apple.com/iphone) основана на OS X, но оптимизирована для мобильных устройств. Программисты iPhone применяют инструментарий для разработки в Mac OS X, называемый XCode. В этом наборе — интегрированная среда разработки, инструмент для создания пользовательских интерфейсов, отладчик и т. д. В Mac OS X и iPhone используется один и тот же инструментарий. Основной фреймворк в этой среде называется Cocoa, а его специализированная версия для мобильных устройств и сенсорных экранов — Cocoa touch. Cocoa написан на Objective-C, объектно-ориентированном языке, представляющем собой дополненную версию языка C. Кроме того, этот язык является динамическим, подобно AppleScript, Python или Ruby. При необходимости можно писать программы на этих скриптовых языках, пользуясь Cocoa Bridge.

По-видимому, **Palm** (http://www.palm.com/us) использует на своих аппаратах свойства и Palm OS, и Windows Mobile. Под Palm OS можно работать с Codewarior или с набором программ для разработчика Palm OS, в который входит Eclipse IDE и компилятор gcc. В Palm OS есть эмуляторы, приспособленные для работы с Eclipse IDE и тестирования программ. Palm OS также позволяет программировать на Java с применением стандарта Java ME.

Windows Mobile (http://msdn.microsoft.com/en-us/windowsmobile/default.aspx) переносит все принципы программирования в Windows на мобильную платформу. При программировании для мобильной Windows используются те же наборы инструментов, что и в Visual Studio и DotNet. Будет работать любой язык, предназначенный для среды времени исполнения DotNet. В данном случае подойдет ряд языков, в том числе C#.

BREW (https://brewmobileplatform.qualcomm.com/) — это мобильная платформа для процессоров Qualcomm. Программирование для BREW выполняется при помощи Visual Studio или Eclipse на языках C и C++. Предпочтительная платформа для разработки — Windows, но код будет работать и в устройствах с ARM. Кроме того, при разработке можно использовать инструменты Flash. BREW также поддерживает Java стандарта Java ME.

JavaFX Mobile (http://www.sun.com/software/javafx/mobile/index.jsp) — это разработка Sun, похожая на Microsoft Silverlight, где для создания пользовательских интерфейсов применяется декларативный подход. JavaFX Mobile работает на любой мобильной платформе, поддерживающей Java ME. Несколько лет назад компания

Sun купила SavaJe, разрабатывавшую мобильные ОС на основе Java. Достижения SavaJe вошли в состав JavaFX Mobile. Среда разработки в JavaFX основана на Java. Это может быть среда Netbeans от Sun или свободная среда Eclipse IDE.

Итак, зная об этих достижениях вчерашнего и сегодняшнего дня, какую судьбу можно предсказать для Android? Есть ли у Android коренные преимущества, которые позволят ей закрепиться на мобильном рынке? Каковы будут ее основные конкуренты?

Сравнение Android с другими мобильными ОС

Посмотрим, как описанные нами выше операционные системы подготовлены к будущему.

Будущее Symbian представляется неопределенным, так как его спонсор Nokia уже использует для высокотехнологичных устройств Maemo. ОС Blackberry слишком проприетарна и полагается на Java ME и на ее широкое распространение. Java ME, в свою очередь, завязла в долгом и запутанном процессе стандартизации. Как минимум при беглом анализе мы не нашли серьезных причин, по которым Blackberry могла бы стать влиятельной силой на мобильном рынке.

Moblin, основанная на Linux, пока остается «запасным игроком», так как Intel развивает ее как ОС для планшетов с выходом в Интернет. Поскольку Moblin — это вид Linux, для разработки в ней применяются инструменты Linux, таким образом, в работу с системой не вовлечены специалисты, знающие Apple и Windows. То же, по-видимому, касается и Maemo. Две эти системы лишены той широкой привлекательности, необходимой в сообществе программистов, где сосуществуют Apple и Windows.

Palm, похоже, пошла другим путем. Эта компания прилагает все усилия, чтобы сделать устройство максимально независимым от операционной системы. При поддержке в устройстве нескольких ОС, вероятно, Palm устоит. Но система, по-видимому, будет склоняться в сторону Windows Mobile. По тем же причинам в Palm в будущем может появиться поддержка Android.

Таким образом, на рынке остается только три по-настоящему сильных игрока: Windows Mobile, iPhone OS и Android. Windows Mobile относится сюда, потому что Microsoft обладает богатейшим опытом продажи программ производителям оборудования. Инструментарии, основанные на Dotnet, довольно внушительны, это касается в том числе Silverlight. Хотя на сегодняшний день эти системы кажутся медленными и полными ошибок, со временем они будут только улучшаться и становиться все привлекательнее, так как вычислительная мощность устройств возрастает. Единственное серьезное «но» — удастся ли оптимизировать основной код Windows так быстро, как этого требует рынок.

Apple также располагает превосходным инструментарием. Но связь с Objective-C и необходимость знания платформы Mac OS X, возможно, продолжат снижать количество разработчиков, хотя на рынке уже более 100 000 программ для iPhone. Если коротко — Apple пока остается инновационным движком.

В этой сфере Android имеет как преимущества, так и недостатки. По сравнению с рассмотренными выше ОС, Android — это одна из наиболее простых

и одновременно комплексных платформ, вся система скачивается за один раз. Специалисту удобно начинать программирование в Android. Однако, если говорить о полноте инструментария для разработки, Windows Mobile может составить Android конкуренцию. Но Android базируется на Java, а этот язык популярен среди разработчиков и привлекает в лагерь Android все новые кадры. Выбирая Java в качестве основного языка программирования, приходится жертвовать вычислительной скоростью, особенно это касается игр. Для подобных целей лучше подходит Apple с его явным управлением памятью. Правда, в перспективе Android может справиться с подобной проблемой, работая с неуправляемыми языками.

В итоге гонку выиграет тот, кто максимально упростит труд разработчиков, останется инновационным и в то же время подвижным. Если мы заговорили о подвижности, продемонстрируем ее на примере.

Поддержка HTML 5 и что за этим стоит

Говоря о фреймворках для разработки, в частности WPF (Windows Presentation Framework), Cocoa Touch (фреймворк для разработки пользовательских интерфейсов в Apple), или об Android, мы упускаем еще одну «рабочую лошадку», которая тянет значительную часть программирования на устройстве. Это браузер. Времена, когда браузер просто отображал HTML-страницы, уже в прошлом. Новая парадигма программирования включает как минимум умение работать с JavaScript и манипулировать объектной моделью документа. Мы обсуждали эти возможности в главе 17, когда говорили о Titanium.

Ситуация складывается в пользу активного изучения и внедрения HTML 5, который поддерживает такие функции, как:

- Web Workers (сложное многопоточное программирование);
- видеоэлементы;
- холсты (canvas);
- кэши приложений и база данных;
- геолокация;
- междокументный обмен сообщениями;
- редактируемость контента;
- серверные события.

Технология Web Workers позволяет запускать несколько потоков для выполнения кода. Ранее для решения подобных задач использовались технологии iframes и Ajax. Теперь все встроено в браузер. Для работы с этими концепциями JavaScript предлагает новые объекты.

Видеоэлементы используются для нативного воспроизведения различных форматов видео прямо в браузере, без применения плагинов Flash или Silverlight.

Элемент холст применяется для рисования чего-либо при помощи скриптового языка, например JavaScript. Есть свободно распространяемый инструмент, который называется BeSpin. В нем используются такие холсты, позволяющие программировать «в облаке», при применении веб-страниц.

Кэш приложения обеспечивает офлайновое хранение такого контента, как, например, электронная переписка и т. д.

Геолокация позволяет пользователю определять свое местоположение — и географически, где это возможно, и по IP-адресу.

Междокументный обмен сообщениями обеспечивает безопасное совместное использование данных двумя разными документами, расположенными в разных местах.

Концепция редактируемости контента означает, что HTML браузера должен воспринимать изменения, вносимые пользователем. Это более непосредственная реализация тех же возможностей, которые предоставляются на вики-ресурсах.

Серверные события — это возможность сервера инициировать события в браузерах. В сумме все эти события превращают браузер в типичный фреймворк для программирования, управляемый многими скриптовыми языками. Большое количество таких возможностей уже поддерживается в Chrome, Firefox, Opera и Safari. Кроме того, эти возможности должны появиться и в следующей версии Internet Explorer. Что касается Android, то в документации прямо сказано о поддержке следующих функций:

- API базы данных для клиентских баз данных, работающих с SQL;
- кэша приложений для программ, работающих офлайн;
- геолокационных API для представления информации о местоположении устройства;
- тега `<video>` в полноэкранном режиме.

Поддержка большинства этих возможностей без серьезного ущерба скорости вычислений позволит создавать в Android потрясающие программы. Именно на этом уровне Android может обойти конкурентов, особенно по мере того, как Google продолжит развивать эту платформу.

В заключение этой главы перечислим полезные ресурсы, связанные с Android.

Ресурсы, посвященные Android

Ресурсы, посвященные Android, можно разделить на две группы. В первую войдут основные ресурсы, которые будут полезны разработчикам. Сайты из второй группы помогут вам быть в курсе событий, происходящих с Android.

Основные ресурсы Android

Перечисленные здесь источники посвящены в основном поддержке Android со стороны Google.

Android Developers home page (http://developer.android.com) — с этой страницы начинается работа программиста, занимающегося Android. По мере выхода новых SDK на этом сайте будут появляться соответствующие ссылки.

The Developer's Guide (http://developer.android.com/guide/index.html) — это руководство по разработке в наиболее актуальной версии Android.

Android 2.0 Features (http://developer.android.com/sdk/android-2.0-highlights.html) — здесь дается общий обзор важных функций, поддерживаемых в Android.

Android 2.0 API level changes (сайт http://developer.android.com/sdk/android-2.0.html#api) — по этой ссылке описаны новые API, действующие в версии 2.0.

Android SDK downloads (http://developer.android.com/sdk/index.html) — по этой ссылке можно скачать инструментарии для разработки в Android.

Android Open Source Project (http://source.android.com/) — здесь вы найдете исходный код к Android SDK.

Google I/O Developer Conference (http://code.google.com/events/io/) — на этом сайте содержится информация о проводимых Google конференциях I/O, посвященных разработке ПО, в том числе материалы секций по Android. Вы сможете просмотреть и материалы конференций за прошлые годы.

Git (http://git-scm.com/) — для работы с кодом Android вам понадобится Git, свободно распространяемый инструмент контроля версий, который облегчает работу с крупными распределенными проектами.

The Android Blog (http://android-developers.blogspot.com/) — здесь вашему вниманию предлагаются глубокие и содержательные статьи и комментарии по внутренней организации Android.

Google Android Developers Group (http://groups.google.com/group/android-developers) — это дискуссионная группа, посвященная вопросам разработки в Android.

Android Issues List (http://code.google.com/p/android/issues/list) — здесь собираются сообщения об обнаруженных в Android ошибках и предлагаются способы их исправления. На этом сайте полезно почитать о проблемах, которые могут возникать при разработке в Android.

Сайт Сатьи Коматинени (http://www.satyakomatineni.com) — Сатья Коматинени ведет специальный дневник о своей работе, исследованиях и коде, создаваемом для Android. Ссылки, размещенные на главной странице, позволяют получить доступ к материалам о различных аспектах Android.

Новостные ресурсы, посвященные Android

Чтобы не отставать от быстро изменяющейся ситуации с платформой Android и следить за всеми важными событиями, можете посещать один или несколько новостных сайтов, посвященных Android.

Anddev.org (http://www.anddev.org) — на этом сайте расположены форумы, где публикуются новости, обзоры телефонов и программ Android, рассказывается о проблемах, возникающих при написании кода, а также даются руководства, включающие исходный код. Темы для сайта подобраны правильно, но не все они рассмотрены одинаково подробно.

Androidandme (http://www.androidandme.com) — этот сайт похож на новостной журнал по Android. На нем рассказывается об операторах, телефонах, программах, играх, случаях взлома, дается помощь для начинающих, есть форумы, а также магазин, где можно купить телефоны и аксессуары к ним. Если ядром anddev.org являются форумы, то здесь предпочтение отдается блогам, сгруппированным по категориям. Форумы здесь также имеются. Создается впечатление, что у сайта — внушительная пользовательская аудитория.

Android guys (http://www.androidguys.com) — на этом новостном сайте делается акцент на подкастах. Основные разделы — новости, магазин для покупки программ и аксессуаров и много подкастов. Магазин на этом сайте очень хороший. Кроме того, сайт можно назвать «блогоцентричным» — описание каждого элемента или новость здесь представляет собой запись блога.

Androidauthority (http://www.androidauthority.com/) — еще один новостной сайт. Здесь публикуются новости и обзоры по телефонам и программам Android. На сайте есть видео, магазины, а также специальный раздел по нетбукам. Но этот сайт позиционируется прежде всего как новостной.

Androidcentral (http://www.androidcentral.com) — на этом новостном сайте есть статьи, новости, магазин и форумы.

Резюме

Мы весьма подробно рассмотрели в этой книге операционную систему Android. Все темы были глубоко изучены и сопровождались практическими примерами. В последней главе мы проанализировали мобильный рынок и предположили, какое место может занять Android в сфере мобильного программирования. В конце мы перечислили ресурсы, которые помогут вам всегда быть в курсе событий, происходящих с Android.

Благодарим вас за то, что дали нам возможность поделиться знаниями об Android. Пожалуйста, не стесняйтесь писать нам, если у вас возникнут вопросы и вы захотите нам их задать. Связаться можно с любым из авторов по следующим адресам:

- Сейд Хашими: hashimisayed@gmail.com;
- Сатья Коматинени: satya.komatineni@gmail.com;
- Дэйв Маклин: davemac327@gmail.com.

Хашими С., Коматинени С., Маклин Д.

Разработка приложений для Android

Перевод с английского О. Сивченко

Заведующий редакцией	*К. Галицкая*
Ведущий редактор	*Е. Каляева*
Художник	*А. Татарко*
Корректор	*Е. Павлович*
Верстка	*Г. Блинов*

ООО «Мир книг», 198206, Санкт-Петербург, Петергофское шоссе, 73, лит. А29.

Налоговая льгота — общероссийский классификатор продукции ОК 005-93, том 2; 95 3005 — литература учебная.

Подписано в печать 05.04.11. Формат 70×100/16. Усл. п. л. 59,340. Тираж 2000. Заказ № 164.

Отпечатано в ГП ПО «Псковская областная типография».
180004, г. Псков, ул. Ротная, 34.



Благодаря этому практическому руководству вы научитесь создавать приложения для устройств на базе ОС Android (мобильных телефонов, планшетных компьютеров, нетбуков, смартбуков), пользуясь новейшими инструментами разработки. Помимо основных вопросов и методик написания программ для Android в книге рассмотрены более сложные темы, в частности создание пользовательских 3D-компонентов, работа с OpenGL и сенсорными экранами, в том числе обработка жестов. Вы узнаете об интегрированных в Android функциях локального и глобального поиска, о внедрении функции машинного перевода Google, о функциях синтеза речи.

Кроме подробного теоретического материала в книге содержатся практические рекомендации от профессионалов и примеры готовых работающих приложений. Вы получите все необходимые знания и навыки для написания приложений любой сложности!

Тема:	**Программирование. Языки и среды разработки**	**Уровень пользователя:**	**опытный**

Apress® SOURCE CODE ONLINE
www.apress.com



ПИТЕР®

ISBN: 978-5-459-00530-1
9 785459 005301

Заказ книг:
197198, Санкт-Петербург, а/я 127, тел.: (812) 703-73-74, postbook@piter.com
61093, Харьков-93, а/я 9130, тел.: (057) 758-41-45, 751-10-02, piter@kharkov.piter.com
www.piter.com — вся информация о книгах и веб-магазин